Dokumentacja Bazy Danych

Systemy Baz Danych 2023/2024 – projekt

Weronika Wojtas, Wiktor Warzecha, Radosław Rolka

1. Opis funkcji systemu z podziałem na użytkowników

1.1. Administratorzy

1.1.1. Starszy administrator

- Nadawanie pozostałych uprawnień
- Tworzenie nowych roli
- Restart systemu
- Wgląd do wszystkich raportów
- Dostęp do wszystkich widoków i funkcji
- Zmiana dostępności webinarów dla użytkowników
- Wgląd do wszystkich tabel z możliwością ich modyfikacji

1.1.2. Młodszy administrator

- Nadawanie uprawnień: koordynator, prowadzący, tłumacz
- Dostęp do wszystkich widoków i funkcji
- Dostęp do wszystkich tabel
- Możliwość tworzenia nowych widoków, oraz funkcji, oraz modyfikacji już istniejących

1.1.3. System

- Przypisywanie webinaru: Przypisuje webinar do wybranego wykładowcy, który może na platformie udostępniać ekran a webinar widnieje w jego kalendarzu
- Rejestracja użytkownika (zapisywanie emailu/hasła, Imienia, Nazwiska, Adresu korespondencyjnego)
- Logowanie użytkownika po podaniu emailu/hasła
- Przypisywanie kursu: przypisuje kurs razem z modułami do wybranego wykładowcy
- Określanie kursu - jeśli kurs posiada chociaż jeden moduł stacjonarny ustawia limit obecności na najmniejszą ilość pojemności sali spośród wszystkich w modułach
- Zaznaczanie obecności modułów asynchronicznych na podstawie obejrzenia nagrania
- Przypisywanie prowadzącym spotkań na studiach
- Przypisywanie instruktorom praktyk
- Generowanie listy raportów finansowych
- Generowanie listy dłużników
- Generowanie listy osób zapisanych na wydarzenia (Imię, Nazwisko, czy wydarzenie stacjonarne/zdalne) - dostęp do wszystkich wydarzeń
- Generowanie listy obecności na wydarzeniu (Data, Imię, Nazwisko, obecny/nieobecny) - dostęp do wszystkich wydarzeń

- Udostępnia użytkownikom przedmioty po ich opłaceniu z odpowiednimi oznaczeniami, płatność całkowita/częściowa, data następnej płatności (webinary, data aktualna, kursy 3 dni przed rozpoczęciem lub aktualna jeśli cała kwota z góry, studia 3 dni przed zjazdem), następna kwota do zapłacenia lub 0 w przypadku jej braku
- Blokuje widok webinarów po upłynięciu 30 dni od daty udostępnienia webinaru lub modułu kursu asynchronicznego
- Generowanie raportu bilokacji - lista osób, które są zapisane na co najmniej dwa przyszłe wydarzenia kolidujące czasowo
- Ustalanie kosztu każdego spotkania na studium (koszt reszty/ilość spotkań)
- Ustawianie limitu studiów na podstawie najmniejszego z limitów miejsc spotkań.
- Po zapisaniu się słuchaczy na studium ustawianie limitu miejsc na jakie mogą zapisać się osoby z zewnątrz wzór:
  limit miejsc na spotkanie - ilość słuchaczy zapisanych na studium
- Zliczanie oceny z egzaminu zdalnego na podstawie liczby %
- Podsumowywanie obecności po zakończonym kursie - zliczanie obecności każdego użytkownika i wysyłanie powiadomienia o pozytywnie/negatywnie zakończonym kursie
- Podsumowywanie studium - podsumowanie obecności spotkań - powyżej 80%, podsumowanie obecności praktyk - 100%, podsumowanie zaliczenia egzaminu - ocena co najmniej 3, wysłanie powiadomienia o pozytywnie/negatywnie zakończonym studium
- Przydzielanie odrabiającym użytkownikom przedmiotów ze statusem opłacone
- Rejestracja użytkownika - dodawanie nowych użytkowników

1.2. Pracownicy

1.2.1. Dyrektor

- Wgląd we wszystkie raporty
- Dostęp do niektórych procedur (policzenie ilości studentów na studiach)
- Odraczanie płatności - udostępnienie użytkownikowi przedmiotu i ustalenie ostatecznej odroczonej daty płatności)

1.2.2. Koordynator (webinaru/studium/kursu)

- Dodawanie webinaru: forma webinaru - płatny/darmowy (cena jeśli płatny), data i godzina, nazwa prowadzącego, język wykładowy, nazwa tłumacza jeśli webinar tego wymaga
- Dodawanie kursu: Cena kursu dzieląca się na zaliczka/reszta, prowadzący kursu, język wykładowy, tłumacz jeśli potrzeba, dodanie modułów.
- Dodawanie modułów do kursów: Rodzaj modułu (stacjonarny, synchroniczny, asynchroniczny).
  - Stacjonarny: data i godzina, sala,

- Synchroniczny: data i godzina
- Asynchroniczny: data i godzina dodania nagrania

- Dodawanie studium: dodawanie sylabusa, dziedziny tematyki, dodawanie harmonogramu studiów z rozróżnieniem zwykłych spotkań, oraz praktyk, ostatnie dwa spotkania przeznaczone są na egzamin, dodanie ceny za całe studia (zaliczka, oraz reszta) dodanie ceny za pojedyncze spotkanie.
- Dodawanie spotkań do studiów, spotkanie musi mieć określoną formę online/stacjo, limit miejsc, prowadzącego, język, tłumacza jeśli jest to wymagane, datę i godzinę, salę jeśli spotkanie jest stacjonarne
- Dodawanie praktyk, praktyki trwają 2 tyg ciągiem po 8 godzin dziennie, muszą mieć określoną datę rozpoczęcia, godzinę, oraz miejsce. Praktyki każdego dnia odbywają się o tej samej godzinie, oraz w tym samym miejscu. Praktyki prowadzi instruktor.
- Możliwość zmiany miejsca, oraz daty, oraz prowadzącego spotkania na studium

1.2.3. Prowadzący/Instruktor

- Zaznaczanie obecności na zajęciach na liście
- Wgląd w raporty listy obecności osób zapisanych na przypisane do nich wydarzenie
- Wgląd w raport listy osób które uczestniczyły w jego wydarzeniu
- Dodawanie plików do swojego wydarzenia (kursu/studium)
- Dodawanie oceny z egzaminu ze studium (jeśli egzamin odbył się stacjonarnie)

1.2.4. Tłumacz

- Tłumaczenie kursów

1.3. Klienci

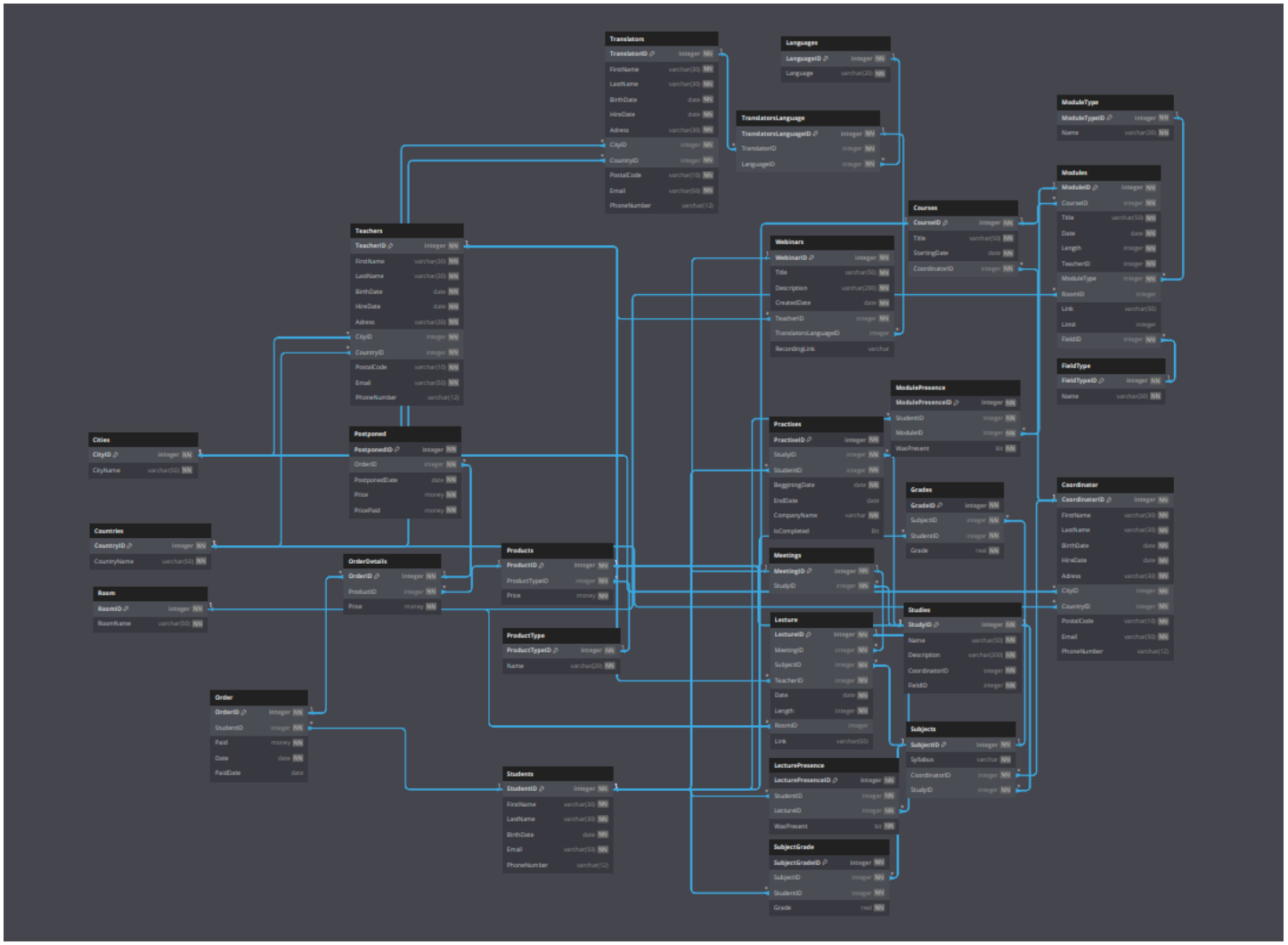
1.3.1. Użytkownik zalogowany

- Kupno produktów

1.3.2. Użytkownik zapisany na studium

- Odrabianie studium na podstawie zapisania się na kurs/spotkanie studyjne w tej samej dziedzinie tematyki, za takie spotkanie użytkownik nie musi płacić

2. Diagram


Translators
TranslatorID integer NN
FirstName varchar(30) NN
LastName varchar(30) NN
BirthDate date NN
HireDate date NN
Adress varchar(30) NN
CityID integer NN
CountryID integer NN
PostalCode varchar(10) NN
Email varchar(50) NN
PhoneNumber varchar(12)
Languages
LanguageID integer NN
Language varchar(20) NN
TranslatorsLanguage
TranslatorsLanguageID integer NN
TranslatorID integer NN
LanguageID integer NN
ModuleType
ModuleTypeID integer NN
Name varchar(20) NN
Modules
ModuleID integer NN
CourseID integer NN
Title varchar(50) NN
Date date NN
Length integer NN
TeacherID integer NN
ModuleType integer NN
RoomID integer
Link varchar(50)
Limit integer
FieldID integer NN
Courses
CourseID integer NN
Title varchar(50) NN
StartingDate date NN
CoordinatorID integer NN
Teachers
TeacherID integer NN
FirstName varchar(30) NN
LastName varchar(30) NN
BirthDate date NN
HireDate date NN
Adress varchar(30) NN
CityID integer NN
CountryID integer NN
PostalCode varchar(10) NN
Email varchar(50) NN
PhoneNumber varchar(12)
Webinars
WebinarID integer NN
Title varchar(50) NN
Description varchar(200) NN
CreatedDate date NN
TeacherID integer NN
TranslatorsLanguageID integer
RecordingLink varchar
FieldType
FieldTypeID integer NN
Name varchar(30) NN
ModulePresence
ModulePresenceID integer NN
StudentID integer NN
ModuleID integer NN
WasPresent bit NN
Cities
CityID integer NN
CityName varchar(50) NN
Postponed
PostponedID integer NN
OrderID integer NN
PostponedDate date NN
Price money NN
PricePaid money NN
Practises
PractiseID integer NN
StudyID integer NN
StudentID integer NN
BeggingDate date NN
EndDate date
CompanyName varchar NN
IsCompleted bit
Grades
GradeID integer NN
SubjectID integer NN
StudentID integer NN
Grade real NN
Coordinator
CoordinatorID integer NN
FirstName varchar(30) NN
LastName varchar(30) NN
BirthDate date NN
HireDate date NN
Adress varchar(30) NN
CityID integer NN
CountryID integer NN
PostalCode varchar(10) NN
Email varchar(50) NN
PhoneNumber varchar(12)
Countries
CountryID integer NN
CountryName varchar(50) NN
OrderDetails
OrderID integer NN
ProductID integer NN
Price money NN
Products
ProductID integer NN
ProductTypeID integer NN
Price money NN
Meetings
MeetingID integer NN
StudyID integer NN
Studies
StudyID integer NN
Name varchar(50) NN
Description varchar(200) NN
CoordinatorID integer NN
FieldID integer NN
Room
RoomID integer NN
RoomName varchar(50) NN
ProductType
ProductTypeID integer NN
Name varchar(20) NN
Lecture
LectureID integer NN
MeetingID integer NN
SubjectID integer NN
TeacherID integer NN
Date date NN
Length integer NN
RoomID integer
Link varchar(50)
Subjects
SubjectID integer NN
Syllabus varchar NN
CoordinatorID integer NN
StudyID integer NN
Order
OrderID integer NN
StudentID integer NN
Paid money NN
Date date NN
PaidDate date
LecturePresence
LecturePresenceID integer NN
StudentID integer NN
LectureID integer NN
WasPresent bit NN
Students
StudentID integer NN
FirstName varchar(30) NN
LastName varchar(30) NN
BirthDate date NN
Email varchar(50) NN
PhoneNumber varchar(12)
SubjectGrade
SubjectGradeID integer NN
SubjectID integer NN
StudentID integer NN
Grade real NN

3. Opis zawartości tabel

3.1. Osoby

3.1.1. Użytkownicy (Students)

Tabela zawierająca informacje o wszystkich użytkownikach.

StudentID integer [primary key, not null]	Numer identyfikacyjny	-
FirstName varchar(30) [not null]	Imię	-
LastName varchar(30) [not null]	Nazwisko	-
BirthDate date [not null]	Data urodzenia	Sprawdza, czy rok urodzenia >= 1900
Email varchar(50) [not null]	adres email	Sprawdza, czy email ma @ w środku i czy jest unikalny
PhoneNumber varchar(12)	numer telefonu	Sprawdza, czy składa się z cyfr

```
create table dbo.Students
(
   StudentID   int identity
       primary key,
   FirstName   varchar(30) not null,
   LastName    varchar(30) not null,
   BirthDate   date        not null
       check (datepart(year, [BirthDate]) >= 1900),
   Email       varchar(50) not null
       unique
       check ([Email] like '%@%'),
   PhoneNumber varchar(12)
       check (NOT [PhoneNumber] like '%[^0-9]%')
)
```

3.1.2. Koordynatorzy (Coordinator)

Tabela zawierająca informacje o wszystkich koordynatorach.

CoordinatorID integer [primary key, not null]	Numer identyfikacyjny	
FirstName varchar(30) [not null]	Imię	

LastName varchar(30) [not null]	Nazwisko	
BirthDate date [not null]	Data urodzenia	Sprawdza, czy rok urodzenia >= 1900
HireDate date [not null]	Data zatrudnienia	
Address varchar(30) [not null]	Adres	
CityID integer [not null]	Numer identyfikacyjny miasta	
CountryID integer [not null]	Numer identyfikacyjny kraju	
PostalCode varchar(10) [not null]	Kod pocztowy	
Email varchar(50) [not null]	Email	Sprawdza, czy email ma @ w środku i czy jest unikalny
PhoneNumber varchar(12)	Numer telefonu	Sprawdza, czy składa się z cyfr

```
create table dbo.Coordinator
(
   CoordinatorID int identity
       primary key,
   FirstName     varchar(30) not null,
   LastName      varchar(30) not null,
   BirthDate     date        not null
       check (datepart(year, [BirthDate]) >= 1900),
   HireDate      date        not null,
   Adress        varchar(30) not null,
   CountryCityID int         not null
       references dbo.CountryCity,
   PostalCode    varchar(10) not null,
   Email         varchar(50) not null
       unique
       check ([Email] like '%@%'),
   PhoneNumber   varchar(12)
       check (NOT [PhoneNumber] like '%[^0-9]%')
)
```

3.1.3. Nauczyciele (Teachers)

Tabela zawiera informacje o wszystkich nauczycielach.

TeacherID integer [primary key, not null]	Numer identyfikacyjny	
FirstName varchar(30) [not null]	Imię	

LastName varchar(30) [not null]	Nazwisko	
BirthDate date [not null]	Data urodzenia	sprawdza czy data urodzenia >1900
HireDate date [not null]	Data zatrudnienia	domyślnie - dzisiejsza data
Address varchar(30) [not null]	Adres	
CountryCityID int [not null]	Numer identyfikacyjny pary kraju i miasta	
PostalCode varchar(10) [not null]	Kod pocztowy	
Email varchar(50) [not null]	Email	sprawdza czy unikalny, sprawdza czy ma w sobie @
PhoneNumber varchar(12)	Numer telefonu	sprawdza czy numer telefonu składa się z cyfr 0-9

```
create table dbo.Teachers
(
   TeacherID     int identity
       primary key,
   FirstName     varchar(30) not null,
   LastName      varchar(30) not null,
   BirthDate     date        not null
       check (datepart(year, [BirthDate]) >= 1900),
   HireDate      date default getdate(),
   Adress        varchar(30) not null,
   CountryCityID int         not null
       references dbo.CountryCity,
   PostalCode    varchar(10) not null,
   Email         varchar(50) not null
       unique
       check ([Email] like '%@%'),
   PhoneNumber   varchar(12)
       check (NOT [PhoneNumber] like '%[^0-9]%')
)
```

3.1.4. Tłumacze (Translators)

Tabela zawiera informacje o wszystkich Tłumaczach.

TranslatorID integer [primary key, not null]	Numer identyfikacyjny	
FirstName varchar(30) [not null]	Imię	
LastName varchar(30) [not null]	Nazwisko	
BirthDate date [not null]	Data urodzenia	
HireDate date [not null]	Data zatrudnienia	domyślnie dzień dzisiejszy

Address varchar(30) [not null]	Adres	
CountryCItyID int [not null]	Numer identyfikacyjny pary kraju i miasta	
PostalCode varchar(10) [not null]	Kod pocztowy	
Email varchar(50) [not null]	Email	unikalny, musi zawierać '@'
PhoneNumber varchar(12)	Numer telefonu	sprawdza czy składa się z cyfr 0-9

```
create table dbo.Translators
(
   TranslatorID  int identity
       primary key,
   FirstName     varchar(30) not null,
   LastName      varchar(30) not null,
   BirthDate     date        not null
       check (datepart(year, [BirthDate]) >= 1900),
   HireDate      date default getdate(),
   Adress        varchar(30) not null,
   CountryCityID int         not null
       references dbo.CountryCity,
   PostalCode    varchar(10) not null,
   Email         varchar(50) not null
       unique
       check ([Email] like '%@%'),
   PhoneNumber   varchar(12)
       check (NOT [PhoneNumber] like '%[^0-9]%')
)
```

3.2. Webinary

3.2.1. Webinary (Webinars)

Lista wszystkich webinarów oraz informacji o nich.

WebinarID integer [primary key, not null]	Numer identyfikacyjny	
Title varchar(50) [not null]	Tytuł	
Description varchar(200) [not null]	Opis	
CreatedDate date [not null]	Data utworzenia	domyślnie dzień dzisiejszy, rok musi być conajmniej 2000
TeacherID integer [not null]	Numer identyfikacyjny nauczyciela	
TranslatorsLanguageID integer	Numer identyfikacyjny języka tłumacza	

RecordingLink varchar	Link do nagrania	

```
 create table dbo.Webinars
(
   WebinarID             int                              not
null
       primary key
       references dbo.Products,
   Title                 varchar(50)                      not
null,
   Description           varchar(200)                     not
null,
   CreatedDate           date          default getdate() not
null,
   TeacherID             int                              not
null
       references dbo.Teachers,
   TranslatorsLanguageID int           default NULL
       references dbo.TranslatorsLanguage,
   RecordingLink         nvarchar(255) default NULL
)
```

3.2.2. Tłumacz i język tłumaczenia (TranslatorsLanguage)

Wszystkie pary tłumaczy oraz języków, z którymi pracują,

TranslatorsLanguageID integer [primary key, not null]	Numer identyfikacyjny
TranslatorID integer [not null]	Numer identyfikacyjny tłumacza
LanguageID integer [not null]	Numer identyfikacyjny języka

```
create table dbo.TranslatorsLanguage
(
   TranslatorsLanguageID int identity
       primary key,
   TranslatorID          int not null
       references dbo.Translators,
   LanguageID            int not null
       references dbo.Languages
)
```

3.2.3. Języki tłumaczenia (Languages)

Lista dostępnych języków webinarów.

LanguageID integer [primary key, not null]	Numer identyfikacyjny języka
Language varchar(20) [not null]	Język

```
create table dbo.Languages
(
```

```
    LanguageID int identity
        primary key,
    Language   varchar(20) not null
)
```

3.3. Kursy

3.3.1. Kursy (Courses)

Lista wszystkich kursów oraz informacji o nich.

CourseID integer [primary key, not null]	Numer identyfikacyjny
Title varchar(50) [not null]	Tytuł
StartingDate date [not null]	Data rozpoczęcia
CoordinatorID integer [not null]	Numer identyfikacyjny koordynatora

```
create table dbo.Courses
(
    CourseID      int         not null
        primary key
        references dbo.Products,
    Title         varchar(50) not null,
    StartingDate  date        not null,
    CoordinatorID int         not null
        references dbo.Coordinator
)
```

3.3.2. Moduły (Modules)

Lista wszystkich modułów oraz informacji o nich.

ModuleID integer [primary key, not null]	Numer identyfikacyjny
CourseID integer [not null]	Numer identyfikacyjny kursu
Title varchar(50) [not null]	Tytuł
Date date [not null]	Data
Length integer [not null, note: ">0"]	Długość
TeacherID integer [not null]	Numer identyfikacyjny nauczyciela
ModuleType integer [not null]	Typ modułu
RoomID integer	Numer identyfikacyjny pokoju
Link varchar(50)	Link
Limit integer	Limit
FieldID integer [not null]	Numer identyfikacyjny dziedziny modułu

```
create table dbo.Modules
(
    ModuleID   int identity
        primary key,
```

```
    CourseID    int          not null
        references dbo.Courses,
    Title       varchar(50) not null,
    Date        datetime     not null,
    Length      int          not null
        check ([Length] > 0),
    TeacherID   int          not null,
    ModuleType int           not null
        references dbo.ModuleType,
    RoomID      int default NULL
        references dbo.Room,
    Link        varchar(50),
    Limit       int
        check ([Limit] > 0),
    FieldID     int          not null
        references dbo.FieldType
)
```

3.3.3. Typy modułów (ModuleType)

Lista wszystkich typów modułów (online synchronicznie, online asynchronicznie, stacjonarnie, hybrydowo).

ModuleTypeID integer [primary key, not null]	Numer identyfikacyjny typu modułu
Name varchar(30) [not null]	Nazwa typu

```
create table dbo.ModuleType
(
    ModuleTypeID int          not null
        primary key,
    Name          varchar(30) not null
)
```

3.3.4. Rodzaj dziedziny (FieldType)

Lista dostępnych dziedzin modułów.

FieldTypeID integer [primary key, not null]	Numer identyfikacyjny typu dziedziny
Name varchar(30) [not null]	Nazwa dziedziny

```
create table dbo.FieldType
(
    FieldTypeID int identity
        primary key,
    Name        varchar(30) not null
)
```

3.3.5. Frekwencja na modułach (ModulePresence)

Tabela zawierająca frekwencję na modułach.

ModulePresenceID Integer [primary key, not null]	Numer identyfikacyjny obecności w module	
StudentID Integer [not null]	Numer identyfikacyjny studenta	
ModuleID Integer [not null]	Numer identyfikacyjny modułu	
WasPresent Bit [not null]	Czy był obecny	domyślnie 0

```
create table dbo.ModulePresence
(
   ModulePresenceID int                                    not
null
       primary key,
   StudentID        int                                    not
null
       references dbo.Students,
   ModuleID         int                                    not
null
       references dbo.Modules,
   WasPresent       bit
       constraint DF_ModulePresence_WasPresent default 0
)
```

3.3.6. Osoby z zaliczonymi kursami (passedCourse)

Lista zawierająca osoby z wyszczególnionymi kursami, które zaliczyły:

PassedID int identity	ID zaliczenia	klucz główny
StudentID int not null	ID studenta	
CourseID int not null	ID kursu	

```
create table dbo.passedCourse
(
   PassedID  int identity
       constraint passedCourse_pk
           primary key,
   StudentID int not null
       constraint passedCourse__StudentID_fk
           references dbo.Students,
   CourseID  int not null
       constraint passedCourse__CourseID_fk
           references dbo.Courses
)
```

3.4. Studia

3.4.1. Oceny końcowe (Grades)

Lista zawierająca oceny końcowe dla poszczególnych przedmiotów od użytkowników.

GradeID integer [primary key, not null]	Numer identyfikacyjny
SubjectID integer [not null]	Numer identyfikacyjny przedmiotu
StudentID integer [not null]	Numer identyfikacyjny studenta
Grade real [not null, note: ">0"]	Ocena

```
create table dbo.Grades
(
   GradeID   int identity
       primary key,
   SubjectID int  not null
       references dbo.Subjects,
   StudentID int  not null
       references dbo.Students,
   Grade     real not null
       check ([Grade] >= 0)
)
```

3.4.2. Studia (Studies)

Lista wszystkich kierunków studiów oraz informacji o nich.

StudyID integer [primary key, not null]	Numer identyfikacyjny
Name varchar(50) [not null]	Nazwa
Description varchar(200) [not null]	Opis
CoordinatorID integer [not null]	Numer identyfikacyjny koordynatora
FieldID integer [not null]	Numer identyfikacyjny pola

```
create table dbo.Studies
(
   StudyID       int          not null
       primary key
       references dbo.Products,
   Name          varchar(50)  not null,
   Description   varchar(200) not null,
   CoordinatorID int          not null
       constraint Studies__CoordinatorID_fk
           references dbo.Coordinator,
   FieldID       int          not null
)
```

3.4.3. Praktyki (Practises)

Lista wszystkich praktyk oraz informacji o nich.

PractiseID integer [primary key, not null]	Numer identyfikacyjny	
StudyID integer [not null]	Numer identyfikacyjny studiów	
StudentID integer [not null]	Numer identyfikacyjny studenta	
BegginingDate date [not null]	Data rozpoczęcia	
EndDate date	Data zakończenia	
CompanyName varchar [not null]	Nazwa firmy	
IsCompleted Bit	Czy ukończono	domyślnie 0

```
create table dbo.Practises
(
   PractiseID    int identity
       primary key,
   StudyID       int           not null
       references dbo.Studies,
   StudentID     int           not null
       references dbo.Students,
   BegginingDate date          not null
       check (datepart(year, [BegginingDate]) >= 2019),
   CompanyName   nvarchar(255) not null,
   IsCompleted   bit default 0,
   EndDate       date
)
```

3.4.4. Zjazdy (Meetings)

Lista wszystkich zjazdów.

MeetingID integer [primary key, not null]	Numer identyfikacyjny
StudyID integer [not null]	Numer identyfikacyjny studiów

```
create table dbo.Meetings
(
   MeetingID int not null
       primary key
       references dbo.Products,
   StudyID   int not null
       references dbo.Studies
)
```

3.4.5. Lista przedmiotów (Subjects)

Lista wszystkich przedmiotów oraz informacji o nich.

SubjectID integer [primary key, not null]	Numer identyfikacyjny
Syllabus varchar [not null]	Program nauczania

CoordinatorID integer [not null]	Numer identyfikacyjny koordynatora
StudyID integer [not null]	Numer identyfikacyjny studiów

```
create table dbo.Subjects
(
   SubjectID     int identity
       primary key,
   Syllabus      nvarchar(255) not null,
   CoordinatorID int           not null
       references dbo.Coordinator,
   StudyID       int           not null
       references dbo.Studies
)
```

3.4.6. Lista ocen cząstkowych (SubjectGrade)

Lista wszystkich ocen cząstkowych dla poszczególnych użytkowników.

SubjectGradeID integer [primary key, not null]	Numer identyfikacyjny
SubjectID integer [not null]	Numer identyfikacyjny przedmiotu
StudentID integer [not null]	Numer identyfikacyjny studenta
Grade real [not null, note: ">0"]	Ocena

```
create table dbo.SubjectGrade
(
   SubjectGradeID int identity
       primary key,
   SubjectID      int  not null
       references dbo.Subjects,
   StudentID      int  not null
       references dbo.Students,
   Grade          real not null
       check ([Grade] > 0)
)
```

3.4.7. Lista zajęć (Lecture)

Lista wszystkich zajęć oraz informacji o nich.

LectureID integer [primary key, not null]	Numer identyfikacyjny
MeetingID integer [not null]	Numer identyfikacyjny spotkania
SubjectID integer [not null]	Numer identyfikacyjny przedmiotu
TeacherID integer [not null]	Numer identyfikacyjny nauczyciela
Date date [not null]	Data
Length integer [not null, note: ">0"]	Długość
RoomID integer	Numer identyfikacyjny pokoju
Link varchar(50)	Link

```
create table dbo.Lecture
(
   LectureID int      not null
       primary key,
   MeetingID int      not null
       references dbo.Meetings,
   SubjectID int      not null
       references dbo.Subjects,
   TeacherID int      not null
       references dbo.Teachers,
   Date      datetime not null,
   Length    int      not null,
   RoomID    int
       references dbo.Room,
   Link      varchar(50)
)
```

3.4.8. Frekwencja na zajęciach (LecturePresence)

Tablica zawierająca obecność użytkowników na poszczególnych zajęciach.

LecturePresenceID integer [primary key, not null]	Numer identyfikacyjny obecności na wykładzie	
StudentID integer [not null]	Numer identyfikacyjny studenta	
LectureID integer [not null]	Numer identyfikacyjny wykładu	
WasPresent bit [not null]	Czy był obecny	domyślnie 0

```
create table dbo.LecturePresence
(
   LecturePresenceID int                                    not
null
       primary key,
   StudentID         int                                    not
null
       references dbo.Students,
   LectureID         int                                    not
null
       references dbo.Lecture,
   WasPresent        bit
       constraint DF_LecturePresence_WasPresent default 0
)
```

3.5. Kupowanie

3.5.1. Płatności odroczone (Postponed)

Tablica zawierająca użytkowników z odroczoną płatnością oraz informacje o tym.

PostponedID integer [primary	Numer identyfikacyjny	

key, not null]		
OrderID integer [not null]	Numer identyfikacyjny zamówienia	
PostponedDate date [not null]	Data odroczenia	domyślnie dzień dzisiejszy
Price money [not null]	Cena	musi być >0
PricePaid money [not null]	Zapłacona cena	musi być >=0

```
create table dbo.Postponed
(
   PostponedID   int   not null
       primary key,
   OrderID       int   not null
       references dbo.OrderDetails,
   PostponedDate date default getdate(),
   Price         money not null
       check ([Price] > 0),
   PricePaid     money not null
)
```

3.5.2. Lista produktów (Products)

Lista wszystkich form kształcenia oraz ich cena.

ProductID integer [primary key, not null]	Numer identyfikacyjny	
ProductTypeID integer [not null]	Numer identyfikacyjny typu produktu	
Price money [not null]	Cena	musi być >0

```
create table dbo.Products
(
   ProductID     int identity
       primary key,
   ProductTypeID int   not null
       references dbo.ProductType,
   Price         money not null
       check ([Price] > 0)
)
```

3.5.3. Kategoria produktu (ProductType)

Kategorie produktów (Webinar, Kurs, Meeting, Studia)

ProductTypeID integer [primary key, not null]	Numer identyfikacyjny
Name varchar(20) [not null]	Nazwa

```
create table dbo.ProductType
(
   ProductTypeID int identity
```

```
        primary key,
    Name            varchar(20) not null
)
```

3.5.4. Zamówienia (Order)

Lista zamówień użytkowników oraz informacje na ten temat.

OrderID integer [primary key, not null]	Numer identyfikacyjny zamówienia	
StudentID integer [not null]	Numer identyfikacyjny studenta	
Paid money [not null, note]	Zapłacona suma	
Date date [not null]	Data zamówienia	
PaidDate date	Data zapłaty	domyślnie dzień dzisiejszy

```
create table dbo.[Order]
(
    OrderID   int identity
        primary key,
    StudentID int   not null
        references dbo.Students,
    Paid      money not null,
    Date      date  not null
        check (datepart(year, [Date]) >= 2019),
    PaidDate  date default getdate()
)
```

3.5.5. Szczegóły zamówienia (OrderDetails)

Szczegóły zamówienia wybranej formy kształcenia.

OrderID integer [primary key, not null]	Numer identyfikacyjny	
ProductID integer [not null]	Numer identyfikacyjny produktu	
Price money [not null]	Cena	musi być >0

```
create table dbo.OrderDetails
(
    OrderDetailsID int identity
        primary key,
    OrderID        int   not null
        references dbo.[Order],
    ProductID      int   not null
        references dbo.Products,
    Price          money not null
        check ([Price] >= 0)
)
```

3.6. Inne

3.6.1. Słownik Miast (Cities)

Zbiór miast.

CityID integer [primary key, not null]	Numer identyfikacyjny miasta
CityName varchar (50) [not null]	Nazwa miasta

```
create table dbo.Cities
(
   CityID   int identity
       primary key,
   CityName varchar(50) not null
)
go
```

3.6.2. Słownik krajów (Countries)

Zbiór krajów.

CountryID integer [primary key, not null]	Numer identyfikacyjny kraju
CountryName varchar (50) [not null]	Nazwa kraju

```
create table dbo.Countries
(
   CountryID   int identity
       primary key,
   CountryName varchar(50) not null
)
```

3.6.3. Słownik miast i krajów (CountryCity)

Lista zawierająca połączenia par miast i krajów

CountryCityID	ID pary
CountryID	ID kraju
CityID	ID miasta

```
create table dbo.CountryCity
(
   CountryCityID int not null
       constraint CountryCity_pk
           primary key,
   CountryID     int
       constraint CountryCity__CountryID_fk
           references dbo.Countries,
   CityID        int
       constraint CountryCity__CityID_fk
```

```
            references dbo.Cities
)
```

3.6.4. Słownik pomieszczeń (Room)

Zbiór pomieszczeń placówki.

RoomID integer [primary key, not null]	Numer identyfikacyjny pokoju
RoomName varchar(50) [not null]	Nazwa pokoju

```
create table dbo.Room
(
   RoomID   int identity
       primary key,
   RoomName varchar(50) not null
)
```

4. Widoki

4.1. Raporty finansowe

Tabela zawierająca jaki przychód wygenerował poszczególny webinar/kurs/studium.

ProductID	Numer identyfikacyjny produktu
Category	Forma kształcenia
Name	Nazwa wydarzenia
Income	Wygenerowany przychód

```
alter view dbo.[Raporty finansowe] as
   SELECT P.ProductID, PT.Name as [Category], W.Title as [Name],
ROUND(SUM(Price), 2) as [Income]
   FROM Products AS P
            inner join Webinars as W on P.ProductID = W.WebinarID
            left join ProductType as PT on P.ProductTypeID =
PT.ProductTypeID
   WHERE P.ProductID IN (SELECT od.ProductID
                         FROM OrderDetails AS od
                         WHERE od.ProductID NOT IN (SELECT
m.MeetingID
                                                    FROM Meetings
AS m))
   GROUP BY P.ProductID, PT.Name, W.Title
   UNION
   SELECT P.ProductID, PT.Name, S.Name, ROUND(SUM(Price), 2)
   FROM Products AS P
            inner join Studies as S on P.ProductID = S.StudyID
            left join ProductType as PT on P.ProductTypeID =
PT.ProductTypeID
   WHERE P.ProductID IN (SELECT od.ProductID
                         FROM OrderDetails AS od
                         WHERE od.ProductID NOT IN (SELECT
m.MeetingID
                                                    FROM Meetings
AS m))
   GROUP BY P.ProductID, PT.Name, S.Name
   UNION
   SELECT P.ProductID, PT.Name, C.Title, ROUND(SUM(Price), 2)
   FROM Products AS P
            inner join dbo.Courses C on P.ProductID = C.CourseID
            left join ProductType as PT on P.ProductTypeID =
PT.ProductTypeID
   WHERE P.ProductID IN (SELECT od.ProductID
                         FROM OrderDetails AS od
                         WHERE od.ProductID NOT IN (SELECT
m.MeetingID
                                                    FROM Meetings
AS m))
   GROUP BY P.ProductID, PT.Name, C.Title
   UNION
   SELECT P.ProductID, PT.Name, 'Meeting' + CAST(ProductID as
VARCHAR), ROUND(SUM(Price), 2)
   FROM Products AS
            inner join Meetings as M on P.ProductID = M.MeetingID
            left join ProductType as PT on P.ProductTypeID =
PT.ProductTypeID
   WHERE P.ProductID IN (SELECT od.ProductID
```

```
                        FROM OrderDetails AS od
                        WHERE od.ProductID NOT IN (SELECT
m.MeetingID
                                                   FROM Meetings
AS m))
   group by P.ProductID, PT.Name
go
```

4.2. Lista “dłużników”

Tabela z ID użytkowników, którzy nie zapłacili wymaganej sumy w ustalonym terminie.

StudentID	Numer identyfikacyjny użytkownika
FirstName	Imię użytkownika
LastName	Nazwisko użytkownika
deficit	zaległe płatności

```
CREATE view dbo.[Lista dluznikow] as
   SELECT S.StudentID,
          S.FirstName,
          S.LastName,
          D.deficit
   FROM Students S
            INNER JOIN
        (SELECT O.StudentID, SUM(OD.Price - O.Paid) AS 'deficit'
         FROM [Order] O
                  INNER JOIN OrderDetails OD on O.OrderID =
OD.OrderID
                  INNER JOIN Products P on OD.ProductID =
P.ProductID
                  INNER JOIN ProductType PT on P.ProductTypeID =
PT.ProductTypeID
                  left JOIN Webinars W on P.ProductID = W.WebinarID
                  left JOIN Studies ST on P.ProductID = ST.StudyID
                  left JOIN Courses C on P.ProductID = C.CourseID
         WHERE O.Paid < OD.Price
           and ((PT.Name = 'Webinar' and W.CreatedDate > GETDATE())
or
                (PT.Name = 'Course' and C.StartingDate >
DATEADD(day, 3, GETDATE())) or
             --(PT.Name = 'Study'   and 1) or
                (PT.Name = 'Meeting' and (SELECT MIN(L.Date)
                                          FROM Lecture L
```

```
                                                    INNER JOIN
Meetings M on L.MeetingID = M.MeetingID
                                                WHERE M.MeetingID =
P.ProductID) > DATEADD(day, 3, GETDATE()))
             )
         GROUP BY O.StudentID) D on D.StudentID = S.StudentID
go
```

4.3. Raport liczby zapisanych osób na przyszłe wydarzenia

Lista przyszłych wydarzeń wraz z ilością zapisanych na nie osób.

ProductID	Numer identyfikacyjny produktu
Title	Nazwa wydarzenia
Type	Typ wydarzenia (Online/Stacjonarnie/…)
Count	Liczba zapisanych osób

```
CREATE view dbo.[Osoby na wydarzeniach] as
   SELECT P.ProductID, W.Title, 'Online' as 'Type', Count(*) as
'Count'
   FROM Products P
            INNER JOIN Webinars W ON P.ProductID = W.WebinarID
            Inner Join OrderDetails OD ON P.ProductID =
OD.ProductID
   WHERE CreatedDate > getdate()
   GROUP BY P.ProductID, W.Title
   UNION
   SELECT P.ProductID, M.Title, MT.Name, Count(*)
   FROM Products P
            INNER JOIN Courses C ON P.ProductID = C.CourseID
            Inner Join OrderDetails OD ON P.ProductID =
OD.ProductID
            Inner Join Modules M ON C.CourseID = M.CourseID
            INNER JOIN ModuleType MT ON M.ModuleType =
MT.ModuleTypeID
   WHERE C.StartingDate > getdate()
   GROUP BY P.ProductID, M.Title, MT.Name
   UNION
   SELECT P.ProductID,
          'Lecture' + CAST(L.LectureID AS VARCHAR) AS LectureID,
          CASE
              WHEN L.Link IS NULL THEN 'Offline'
              ELSE 'Online'
              END                                   AS LectureType,
          Count(*)
```

```
   FROM Products P
            INNER JOIN OrderDetails OD ON P.ProductID =
OD.ProductID
            INNER JOIN Meetings M ON P.ProductID = M.MeetingID
            INNER JOIN Lecture L ON M.MeetingID = L.MeetingID
   WHERE L.Date > getdate()
   GROUP BY P.ProductID, L.LectureID, L.Link
go
```

4.4. Ogólny raport dotyczący frekwencji

Zbiór przeszłych wydarzeń z wyszczególnioną frekwencją, ilością zapisanych osób oraz wyliczonym procentem obecnych.

ID	Numer identyfikacyjny wydarzenia
Type	Kategoria wydarzenia (Moduł/Zajęcia)
NumberOfWasPresents	Liczba osób obecnych
TotalParticipants	Wszyscy zapisani na wydarzenie
Percentage	Procent osób obecnych

```
CREATE view dbo.[Frekwencja na zakonczonych wydarzeniach - done] as
   SELECT MP.ModuleID
AS ID,
           'Module'
AS Type,
           COUNT(CASE WHEN MP.WasPresent = 1 THEN 1 END)
AS NumberOfWasPresents,
           COUNT(DISTINCT MP.StudentID)
AS TotalParticipants,
           COUNT(CASE WHEN MP.WasPresent = 1 THEN 1 END) /
CAST(COUNT(*) AS Float) AS Percentage
   FROM ModulePresence MP
   GROUP BY MP.ModuleID

   UNION

   SELECT LP.LectureID
AS ID,
           'Lecture'
AS Type,
           COUNT(CASE WHEN LP.WasPresent = 1 THEN 1 END)
AS NumberOfWasPresents,
           COUNT(DISTINCT LP.StudentID)
AS TotalParticipants,
```

```
          COUNT(CASE WHEN LP.WasPresent = 1 THEN 1 END) /
Cast(COUNT(*) as Float) AS Percentage
   FROM LecturePresence LP
   GROUP BY LP.LectureID
go
```

4.5. Lista obecności dla każdego szkolenia

Lista uczestników dla każdego szkolenia wraz z datą, imieniem, nazwiskiem i informacją o obecności.

Name	Nazwa szkolenia
date	Data szkolenia
StudentID	Numer identyfikacyjny użytkownika
FirstName	Imię użytkownika
LastName	Nazwisko użytkownika
WasPresent	Informacja, czy był obecny

```
CREATE view dbo.[Lista obecnosci dla kazdego szkolenia - done] as
   SELECT 'lecture' + cast(LP.LectureID as varchar)
                  as 'Name',
          L.Date as 'date',
          S.StudentID,
          S.FirstName,
          S.LastName,
          LP.WasPresent
   from LecturePresence LP
            inner join Lecture L on L.LectureID = LP.LectureID
            left join Students S on S.StudentID = LP.StudentID
   union
   SELECT 'module' + cast(MP.ModuleID as varchar)
                  as 'Name',
          M.Date as 'date',
          S.StudentID,
          S.FirstName,
          S.LastName,
          MP.WasPresent
   from ModulePresence MP
            inner join Modules M on M.ModuleID = MP.ModuleID
            left join Students S on S.StudentID = MP.StudentID
```

4.6. Raport bilokacji

Tabela zawierająca listę ID użytkowników, którzy zapisali się na kolidujące ze sobą czasowo szkolenia oraz informacje na temat tych zajęć..

Student	Id studenta
Type	Typ wydarzenia
Product	ID produktu
BeginningDate	Data rozpoczęcia
EndDate	Data zakończenia
Type2	Typ wydarzenia
Product2	ID produktu
BeginningDate2	Data rozpoczęcia drugiego wydarzenia
EndDate2	Data zakończenia drugiego wydarzenia

```
WITH ModulesTable AS (SELECT S.StudentID                    AS
Student,
                             'Module'                       AS
Type,
                             P.ProductID                    AS
Product,
                             M.Date                         AS
BeginningDate,
                             DATEADD(minute, Length, M.Date) AS
EndDate,
                             ModuleID                       AS
ID
                      FROM Students AS S
                               LEFT JOIN [Order] AS O ON
O.StudentID = S.StudentID
                               LEFT JOIN OrderDetails as OD ON
OD.OrderID = O.OrderID
                               LEFT JOIN Products AS P ON
P.ProductID = OD.ProductID
                               LEFT JOIN Courses AS C ON
C.CourseID = P.ProductID
                               LEFT JOIN Modules AS M ON
M.CourseID = C.CourseID
                      WHERE M.ModuleType != 3),
        MeetingsTable AS (SELECT S1.StudentID                  AS
Student,
                             'Meeting'                      AS
Type,
```

```
                                P.ProductID                    AS
Product,
                                L.Date                         AS
BeginningDate,
                                DATEADD(minute, Length, L.Date) as
EndDate,
                                LectureID                      AS
ID
                           FROM Students AS S1
                                  LEFT JOIN [Order] AS O ON
O.StudentID = S1.StudentID
                                  LEFT JOIN OrderDetails as OD ON
OD.OrderID = O.OrderID
                                  LEFT JOIN Products AS P ON
P.ProductID = OD.ProductID
                                  LEFT JOIN Meetings AS M ON
M.MeetingID = P.ProductID
                                  LEFT JOIN Lecture AS L ON
L.MeetingID = M.MeetingID),
        StudiesTable AS (SELECT S2.StudentID                   AS
Student,
                                'Study'                        AS
Type,
                                P.ProductID                    AS
Product,
                                L.Date                         AS
BeginningDate,
                                DATEADD(minute, Length, L.Date) AS
EndDate,
                                LectureID                      AS
ID
                           FROM Students AS S2
                                  LEFT JOIN [Order] AS O ON
O.StudentID = S2.StudentID
                                  LEFT JOIN OrderDetails as OD ON
OD.OrderID = O.OrderID
                                  LEFT JOIN Products AS P ON
P.ProductID = OD.ProductID
                                  LEFT JOIN Studies AS S ON
S.StudyID = P.ProductID
                                  LEFT JOIN Meetings AS M ON
M.StudyID = S.StudyID
                                  LEFT JOIN Lecture AS L ON
L.MeetingID = M.MeetingID),
        UT AS (SELECT T1.Student       AS Student,
                      T1.Type          AS Type,
                      T1.Product       AS Product,
                      T1.BeginningDate AS BeginningDate,
```

```
                    T1.EndDate       AS EndDate,
                    T1.ID            AS ID
             FROM ModulesTable as T1
             UNION
             SELECT T2.Student       AS Student,
                    T2.Type          AS Type,
                    T2.Product       AS Product,
                    T2.BeginningDate AS BeginningDate,
                    T2.EndDate       AS EndDate,
                    T2.ID            AS ID
             FROM MeetingsTable as T2
             UNION
             SELECT T3.Student       AS Student,
                    T3.Type          AS Type,
                    T3.Product       AS Product,
                    T3.BeginningDate AS BeginningDate,
                    T3.EndDate       AS EndDate,
                    T3.ID            AS ID
             FROM StudiesTable as T3)
   SELECT T1.Student,
          T1.Type,
          T1.Product,
          T1.ID,
          T1.BeginningDate,
          T1.EndDate,
          T2.Type          AS Type2,
          T2.Product       AS Product2,
          T2.ID            AS ID2,
          T2.BeginningDate AS BeginningDate2,
          T2.EndDate       AS EndDate2
   FROM UT as T1
            INNER JOIN UT as T2 ON T1.Student = T2.Student
   WHERE ((T1.EndDate >= T2.BeginningDate AND T1.EndDate <=
T2.EndDate) OR
          (T2.EndDate >= T1.BeginningDate AND T2.EndDate <=
T1.EndDate))
     AND T1.BeginningDate > GETDATE()
     AND T2.BeginningDate > GETDATE()
     AND (T1.Product < T2.Product OR (T1.Product = T2.Product AND
T1.ID < T2.ID ))
```

4.7. Harmonogram zajęć dla studentów

Tabela zawierająca id, oraz imię i nazwisko studenta, a także wszystkie zajęcia na które jest on zapisany włącznie z datą.

```
SELECT Students.StudentID,
          Products.ProductID,
          CONCAT(Students.FirstName, ' ', Students.LastName) AS
'Name',
          (CASE
               WHEN ProductType.ProductTypeID = 1 THEN 'Webinar'
               WHEN ProductType.ProductTypeID = 2 THEN 'Module'
               WHEN ProductType.ProductTypeID = 4 OR
ProductType.ProductTypeID = 3 THEN 'Lecture'
               ELSE ''
              END)                                              AS
'Type',
          (CASE
               WHEN ProductType.ProductTypeID = 1 THEN
Webinars.Title
               WHEN ProductType.ProductTypeID = 2 THEN
Modules.Title
               WHEN ProductType.ProductTypeID = 4 THEN Studies.Name
               WHEN ProductTYpe.ProductTypeID = 3 THEN S.name
               ELSE '' END)                                        as
'Title',
          (CASE
               WHEN ProductType.ProductTypeID = 1 THEN
Webinars.CreatedDate
               WHEN ProductType.ProductTypeID = 2 THEN Modules.Date
               WHEN ProductType.ProductTypeID = 4 THEN L.Date
               WHEN ProductType.ProductTypeID = 3 THEN L1.Date
               ELSE NULL
              END)                                              AS
'Data'
   FROM Students
            INNER JOIN [Order] ON Students.StudentID =
[Order].StudentID
            INNER JOIN OrderDetails ON [Order].OrderID =
OrderDetails.OrderDetailsID
            INNER JOIN Products ON OrderDetails.ProductID =
Products.ProductID
            INNER JOIN ProductType ON Products.ProductTypeID =
ProductType.ProductTypeID
            LEFT OUTER JOIN Webinars ON Products.ProductID =
Webinars.WebinarID
            LEFT OUTER JOIN dbo.Courses C ON Products.ProductID =
C.CourseID
            LEFT OUTER JOIN Meetings ON Products.ProductID =
Meetings.MeetingID
            LEFT OUTER JOIN Studies ON Products.ProductID =
Studies.StudyID
```

```
            LEFT OUTER JOIN Meetings M ON Studies.StudyID =
M.MeetingID
            LEFT OUTER JOIN Lecture ON Meetings.MeetingID =
Lecture.MeetingID
            LEFT OUTER JOIN Lecture L ON M.MeetingID = L.MeetingID
            LEFT OUTER JOIN Modules ON C.CourseID =
Modules.CourseID
            LEFT OUTER JOIN Lecture L1 ON Meetings.MeetingID =
L1.MeetingID
            LEFT OUTER JOIN Studies S ON Meetings.StudyID = S.Study
```

4.8. Opis kursu

Tabela zawierająca wszystkie kursy, oraz moduły do nich przypisane z datą odbywania się kursu, oraz trybem (online, synchroniczny, asynchroniczny, stacjonarny), pokój w którym odbywają się zajęcia, link, oraz dziedzinę modułu.

```
SELECT C.CourseID,
          C.Title         as 'Course Title',
          M.Title         as 'Module Name',
          M.Date          as 'Module Date',
          ModuleType.Name as 'Module Type',
          M.RoomID        as 'Room',
          M.Link          as 'Link',
          FieldType.Name  as 'Field'
   FROM Courses as C
            LEFT OUTER JOIN Modules as M ON C.CourseID = M.CourseID
            LEFT OUTER JOIN ModuleType ON M.ModuleType =
ModuleType.ModuleTypeID
            LEFT OUTER JOIN FieldType ON M.FieldID =
FieldType.FieldTypeID
```

4.9. Syllabus studiów

Tabela pokazująca ID wszystkich studiów, ich nazwę, oraz syllabus każdego przypisanego do nich przedmiotu.

```
select Studies.StudyID, Studies.Name as 'Studies name',
Subjects.Syllabus
   FROM studies
            JOIN subjects ON studies.studyid = subjects.studyid
```

4.10. Przedmioty posiadane przez użytkowników

Tabela pokazująca dla studentów posiadających przedmioty każdy zakupiony przez nich przedmiot włącznie z typem przedmiotu

```
SELECT Students.StudentID,
```

```
          CONCAT(Students.FirstName, ' ', Students.LastName) as
'Name',
          ProductType.Name                                   as
'Type',
          (SELECT CASE
                      WHEN ProductType.ProductTypeID = 1 THEN
Webinars.Title
                      WHEN ProductType.ProductTypeID = 2 THEN
C.Title
                      WHEN ProductType.ProductTypeID = 4 THEN
Studies.Name
                      WHEN ProductType.ProductTypeID = 3 THEN ''
                      ELSE ''
                      END AS ProductType)                    as
'Title'
   FROM Students
            INNER JOIN [Order] ON Students.StudentID =
[Order].StudentID
            INNER JOIN OrderDetails ON [Order].OrderID =
OrderDetails.OrderDetailsID
            INNER JOIN Products ON OrderDetails.ProductID =
Products.ProductID
            INNER JOIN ProductType ON Products.ProductTypeID =
ProductType.ProductTypeID
            LEFT OUTER JOIN Webinars ON Products.ProductID =
Webinars.WebinarID
            LEFT OUTER JOIN dbo.Courses C on Products.ProductID =
C.CourseID
            LEFT OUTER JOIN Meetings ON Products.ProductID =
Meetings.MeetingID
            LEFT OUTER JOIN Studies ON Products.ProductID =
Studies.StudyID
```

4.11. Spis pracowników

Widok pokazujący spis pracowników z uwzględnieniem pełnionych funkcji

```
use u_wojtas
go

create view dbo.[Spis pracowników] as
SELECT 'Translator' AS Role, TranslatorID AS ID, FirstName + ' ' +
LastName AS [Imie i Nazwisko] FROM Translators
UNION
SELECT 'Coordinator' AS Role, CoordinatorID AS ID, FirstName + ' '
+ Lastname AS [Imie i Nazwisko] FROM Coordinator
UNION
SELECT 'Teacher' AS Role, TeacherID AS ID, FirstName + ' ' +
Lastname AS [Imie i Nazwisko] FROM Teachers
```

```
go
```

4.12. Praktyki

Widok pokazujący dla wszystkich studentów odbyte praktyki wraz z datą i informacją o ukończeniu, oraz informację ‘Brak praktyk’ jeżeli student do nich nie podszedł.

```
create view dbo.Praktyki as
SELECT S.StudentID, S.FirstName + ' ' + S.LastName AS 'Imie i
Nazwisko', P.CompanyName, P.BegginingDate, P.EndDate, P.IsCompleted
FROM Students AS S
RIGHT JOIN Practises AS P ON S.StudentID = P.StudentID
UNION
SELECT S2.StudentID, S2.FirstName + ' ' + S2.LastName,'Praktyki
nieodbyte', NULL, NULL, NULL FROM Students AS S2
WHERE S2.StudentID NOT IN (SELECT S.StudentID FROM Students AS S
RIGHT JOIN Practises AS P ON S.StudentID = P.StudentID)
```

4.13. Studenci i zdawanie przedmiotów

Widok zapewnia wgląd w aktualny stan zdawalności przedmiotów wśród studentów:

@Email	email studenta
@SubjectID	numer przedmiotu
@Percent	Procent frekwencji
@AverageGrade	średnia z ocen cząstkowych

```
CREATE VIEW dbo.AttendanceAndGradeView
AS
WITH avg_grade AS (
    SELECT
        StudentID,
        SubjectID,
        AVG(Grade) AS AverageGrade
    FROM
        SubjectGrade
    GROUP BY
        StudentID,
        SubjectID
),
attendance AS (
    SELECT
        LP.StudentID,
        L.SubjectID,
        SUM(CAST(LP.WasPresent AS int)) * 1.0 / COUNT(*) AS [Percent]
    FROM
        LecturePresence LP
        INNER JOIN Lecture L ON LP.LectureID = L.LectureID
    GROUP BY
        LP.StudentID,
        L.SubjectID
)
SELECT
    ST.Email,
    A.SubjectID,
    round(A.[Percent],2) AS [Percent],
    round(AG.AverageGrade,2) AS [AverageGrade],
    CASE
        WHEN A.[Percent] >= 0.8 AND AG.AverageGrade >= 3 THEN 'Pass'
        ELSE 'Fail'
    END AS [Status]
FROM
```

```
    Students ST
    INNER JOIN avg_grade AG ON ST.StudentID = AG.StudentID
    INNER JOIN attendance A ON ST.StudentID = A.StudentID;
```

5. Procedury

5.1. AddStudent

Wstawia nowy rekord do Tabeli Student:

@FirstName	Imię użytkownika	napis do 30 znaków
@LastName	Nazwisko użytkownika	napis do 30 znaków
@BirthDate	Data urodzenia	data
@Email	Email	napis do 50 znaków
@PhoneNumber	Numer telefonu	napis do 12 znaków

```
CREATE PROCEDURE AddStudent
    @FirstName varchar(30),
    @LastName varchar(30),
    @BirthDate date,
    @Email varchar(50),
    @PhoneNumber varchar(12)
AS
BEGIN
    SET NOCOUNT ON;

    -- Check if the email is unique before inserting
    IF NOT EXISTS (SELECT 1 FROM Students WHERE Email = @Email)
    BEGIN
        INSERT INTO Students (FirstName, LastName, BirthDate, Email,
PhoneNumber)
        VALUES (@FirstName, @LastName, @BirthDate, @Email,
@PhoneNumber);

        PRINT 'Student added successfully.';
    END
    ELSE
    BEGIN
        PRINT 'Error: Email already exists. Please use a unique
email address.';
    END
END;
```

5.2. AddTeacher

Dodaje nowy rekord do Tabeli Teachers:

@FirstName	Imię Nauczyciela	napis do 30 znaków
@LastName	Nazwisko Nauczyciela	napis do 30 znaków
@BirthDate	Data urodzenia	data
@HireDate	Data zatrudnienia	data, deafult data dzisiejsza
@Address	Adres	napis do 30 znaków
@CityName	Miasto	napis do 50 znaków
@CountryName	Kraj	napis do 50 znaków
@PostalCode	Kod pocztowy	napis do 10 znaków
@Email	Email	napis do 50 znaków
@PhoneNumber	Numer telefonu	napis do 12 znaków

```
CREATE PROCEDURE AddTeacher
  @FirstName varchar(30),
  @LastName varchar(30),
  @BirthDate date,
  @HireDate date,
  @Address varchar(30),
  @City varchar(30),
  @Country varchar(30),
  @PostalCode varchar(10),
  @Email varchar(50),
  @PhoneNumber varchar(12)
AS
BEGIN
  SET NOCOUNT ON;

  IF EXISTS (
      SELECT 1
      FROM Teachers
      WHERE Email = @Email
  )
  BEGIN
      PRINT 'Error: Teacher with the same email already exists.';
      RETURN;
  END

  IF EXISTS (
```

```
      SELECT 1
      FROM Teachers
      WHERE PhoneNumber = @PhoneNumber
  )
   BEGIN
        PRINT 'Error: Teacher with the same phone number already
exists.';
        RETURN;
   END

   IF @BirthDate < 1900-01-01
   BEGIN
       PRINT 'Error: Birth date is invalid.';
       RETURN;
   END

   IF @HireDate < 2019-01-01
   BEGIN
       PRINT 'Error: Hire date is invalid.';
       RETURN;
   END

   SET @City = (SELECT CityID FROM Cities WHERE CityName = @City);
   SET @Country = (SELECT CountryID FROM Countries WHERE
CountryName = @Country);

   IF NOT EXISTS (
       SELECT *
       FROM CountryCity
       WHERE CityID = @City AND CountryID = @Country
   )
   BEGIN
       AddCountryCity @City, @Country;
   END

   DECLARE @CityCountryID int;
   SET @CityCountryID = (SELECT CountryCityID FROM CountryCity
WHERE CityID = @City AND CountryID = @Country);

  INSERT INTO Teachers (FirstName, LastName, BirthDate, HireDate,
Adress, CountryCityID, PostalCode, Email, PhoneNumber)
  VALUES (@FirstName, @LastName, @BirthDate, @HireDate, @Address,
@CityCountryID, @PostalCode, @Email, @PhoneNumber);


  PRINT 'Teacher added successfully.';
END;
GO
```

5.3. AddTranslator

Dodaje nowy rekord do Tabeli Translators:

@FirstName	Imię Tłumacza	napis do 30 znaków
@LastName	Nazwisko Tłumacza	napis do 30 znaków
@BirthDate	Data urodzenia	data
@HireDate	Data zatrudnienia	data
@Address	Adres	napis do 30 znaków
@CityName	Miasto	napis do 50 znaków
@CountryName	Kraj	napis do 50 znaków
@PostalCode	Kod pocztowy	napis do 10 znaków
@Email	Email	napis do 50 znaków
@PhoneNumber	Numer telefonu	napis do 12 znaków

```
CREATE PROCEDURE AddTranslator
  @FirstName varchar(30),
  @LastName varchar(30),
  @BirthDate date,
  @HireDate date,
  @Address varchar(30),
  @City varchar(30),
  @Country varchar(30),
  @PostalCode varchar(10),
  @Email varchar(50),
  @PhoneNumber varchar(12),
   @Language varchar(30)
AS
BEGIN
  SET NOCOUNT ON;

  IF EXISTS (
      SELECT 1
      FROM Translators
      WHERE Email = @Email
  )
  BEGIN
```

```
      PRINT 'Error: Translator with the same email already
exists.';
      RETURN;
  END

  IF EXISTS (
      SELECT 1
      FROM Translators
      WHERE PhoneNumber = @PhoneNumber
  )
   BEGIN
        PRINT 'Error: Translator with the same phone number already
exists.';
        RETURN;
   END

   IF @BirthDate < 1900-01-01
   BEGIN
       PRINT 'Error: Birth date is invalid.';
       RETURN;
   END

   IF @HireDate < 2019-01-01
   BEGIN
       PRINT 'Error: Hire date is invalid.';
       RETURN;
   END

   SET @City = (SELECT CityID FROM Cities WHERE CityName = @City);
   SET @Country = (SELECT CountryID FROM Countries WHERE
CountryName = @Country);

   IF NOT EXISTS (
       SELECT *
       FROM CountryCity
       WHERE CityID = @City AND CountryID = @Country
   )
   BEGIN
       AddCountryCity @City, @Country;
   END


   DECLARE @CityCountryID int;
   SET @CityCountryID = (SELECT CountryCityID FROM CountryCity
WHERE CityID = @City AND CountryID = @Country);

  INSERT INTO Translators (FirstName, LastName, BirthDate,
HireDate, Adress, CountryCityID, PostalCode, Email, PhoneNumber)
```

```
  VALUES (@FirstName, @LastName, @BirthDate, @HireDate, @Address,
@CityCountryID, @PostalCode, @Email, @PhoneNumber);

  DECLARE @TranslatorID int;
   SET @TranslatorID = (SELECT TranslatorID FROM Translators WHERE
Email = @Email);
  AddTranslatorLanguage @TranslatorID, @Language;


  PRINT 'Translator added successfully.';
END;
GO
```

5.4. AddStudy

Dodaje nowy rekord do Tabeli Studies oraz Products:

@StudyName	Nazwa studium	napis do 50 znaków
@StudyDescription	Syllabus studium	napis do 200 znaków
@StudyCoordinatorID	ID koordynatora	integer
@StudyFieldID	ID dziedziny	integer
@Price	Cena studium	money

```
CREATE PROCEDURE AddStudy
  @StudyName varchar(50),
  @StudyDescription varchar(200),
  @StudyCoordinatorID integer,
  @StudyFieldID integer,
  @Price money
AS
BEGIN
  SET NOCOUNT ON;

  IF NOT @StudyCoordinatorID IN (
      SELECT CoordinatorID FROM Coordinator
  )
  BEGIN
      PRINT 'Error: Coordinator does not exist.';
      RETURN;
  END

  IF NOT @StudyFieldID IN (
      SELECT FieldType.FieldTypeID FROM FieldType
```

```
  )
  BEGIN
      PRINT 'Error: StudyField does not exist.';
      RETURN;
  END

  DECLARE @StudyID integer;

  INSERT INTO Products(ProductTypeID, Price)
  VALUES (4, @Price)
  SET @StudyID = SCOPE_IDENTITY();
  INSERT INTO Studies (StudyID, Name, Description, CoordinatorID,
FieldID)
  VALUES (@StudyID, @StudyName, @StudyDescription,
@StudyCoordinatorID, @StudyFieldID);

  PRINT 'Study added successfully.';
END;
```

5.5. BuyStudy

Funkcja pozwala na zakup studium.

@StudentEmail	Email studenta	napis do 50 znaków
@StudyTitle	Nazwa studium	napis do 50 znaków
@OrderID	ID zamówienia w ramach którego jest realizowany zakup	integer
@Paid	Kwota którą płaci kupujący	money

```
CREATE PROCEDURE BuyStudy
   @StudentEmail varchar(50),
   @StudyTitle varchar(50),
   @OrderID int,
   @Paid money = NULL
AS
BEGIN
   SET NOCOUNT ON;
   DECLARE @StudentID int, @StudyID int, @Price money;

       -- Check if the specified student exists
       IF NOT EXISTS (SELECT 1 FROM Students WHERE Email =
@StudentEmail)
       BEGIN
```

```
            PRINT 'Error: Student with email ' + @StudentEmail + '
does not exist.';
        END

        -- Check if the specified study exists
        IF NOT EXISTS (
            SELECT S.StudyID
            FROM Studies AS S
            INNER JOIN Products AS P ON S.StudyID = P.ProductID
            WHERE S.Name = @StudyTitle
        )
        BEGIN
            PRINT 'Error: Webinar with title ' + @StudyTitle + '
does not exist.';
        END

        -- Get StudentID
        SELECT @StudentID = StudentID FROM Students WHERE Email =
@StudentEmail;

        -- Get StudyID and Price
        SELECT @StudyID = S.StudyID, @Price = P.Price
        FROM Studies AS S
        INNER JOIN Products AS P ON S.StudyID = P.ProductID
        WHERE S.Name = @StudyTitle;

        --Check if studies are still available to buy
        DECLARE @StudyBegin date;
        SELECT TOP 1 Date = @StudyBegin FROM Lecture AS L
            WHERE L.MeetingID IN
                (SELECT M.MeetingID FROM Meetings as M
                WHERE M.StudyID = @StudyID)
        ORDER BY Date;

        IF NOT (DATEADD(DAY, 3, @StudyBegin) <= GETDATE())
        BEGIN
            PRINT 'It is too late to sign in for ' + @StudyTitle +
'.';
        END

        --Check limit
        DECLARE @Count int;
        EXEC CountStudentsOnStudy @StudyID, @Count;
        IF NOT (@Count < (SELECT studies_limit FROM Studies WHERE
StudyID = @StudyID))
        BEGIN
            PRINT 'There are too many student signed for' +
@StudyTitle + '.';
```

```
        END

        -- Set @Paid to 0 if it is NULL
        IF @Paid IS NULL SET @Paid = 0;
    -- Start the transaction
    BEGIN TRANSACTION;

    BEGIN TRY
        -- Insert into OrderDetails
        INSERT INTO OrderDetails (OrderID, ProductID, Price)
        VALUES (@OrderID, @StudyID, @Price);

        -- Commit the transaction
        COMMIT;
        PRINT 'Study purchased successfully.';
    END TRY
    BEGIN CATCH
        -- Rollback the transaction if an error occurs
        ROLLBACK;
        PRINT 'Transaction rolled back. An error occurred during the
purchase process.';
    END CATCH;
END;
go

grant execute on dbo.BuyStudy to Student
go
```

5.6. AddWebinar

dodaje nowy produkt - webinar.

@Title	Tytuł webinaru	napis do 50 znaków
@Description	Opis webinaru	napis do 200 znaków
@TeacherEmail	Email prowadzącego	napis do 50 znaków
@Price	Cena	liczba
@CreatedDate	Data webinaru	data. domyślnie dzisiejsza
@Language	Język webinaru (inny niż polski)	napis do 20 znaków, domyślnie brak
@TranslatorEmail	Email Tłumacza	napis do 50 znaków, domyślnie brak

@RecordingLink	Link do webinaru poprzez zewnętrzny serwis	napis do 250 znaków, domyślnie brak

```
CREATE PROCEDURE AddWebinar
    @Title varchar(50),
    @Description varchar(200),
    @TeacherEmail varchar(50),
    @Price money,
    @CreatedDate date = Null,
    @Language varchar(20) = Null,
    @TranslatorEmail varchar(50) = Null,
    @RecordingLink nvarchar(255) = NULL
AS
BEGIN
    SET NOCOUNT ON;

    -- Check if the specified teacher exists
    IF NOT EXISTS (SELECT 1 FROM Teachers WHERE Email =
@TeacherEmail)
    BEGIN
        PRINT 'Error: Teacher ' + @TeacherEmail + ' does not exist.';
        RETURN;
    END

    -- Check if the specified translator exists (if provided)
    IF @TranslatorEmail IS NOT NULL AND NOT EXISTS (
        SELECT 1
        FROM Translators AS T
        INNER JOIN TranslatorsLanguage AS TL ON T.TranslatorID =
TL.TranslatorID
        INNER JOIN Languages AS L ON TL.LanguageID = L.LanguageID
        WHERE T.Email = @TranslatorEmail AND L.Language = @Language
    )
    BEGIN
        PRINT 'Error: Translator ' + @TranslatorEmail + ' for
language ' + @Language + ' does not exist.';
        RETURN;
    END

    -- Insert into Products table
    INSERT INTO Products (ProductTypeID, Price)
    VALUES (
        (SELECT ProductTypeID FROM ProductType WHERE Name =
'Webinar'),
        @Price
    );

    DECLARE @WebinarID int;
```

```
    SET @WebinarID = SCOPE_IDENTITY(); -- Get the identity value of
the recently inserted row

    if @CreatedDate IS NULL
        SET @CreatedDate = GETDATE();

    -- Insert into Webinars table
    INSERT INTO Webinars (WebinarID, Title, Description, CreatedDate,
TeacherID, TranslatorsLanguageID, RecordingLink)
    VALUES (
        @WebinarID,
        @Title,
        @Description,
        @CreatedDate,
        (SELECT TeacherID FROM Teachers WHERE Email = @TeacherEmail),
        (SELECT TL.TranslatorsLanguageID
         FROM TranslatorsLanguage AS TL
         INNER JOIN Translators AS T ON TL.TranslatorID =
T.TranslatorID
         INNER JOIN Languages AS L ON TL.LanguageID = L.LanguageID
         WHERE T.Email = @TranslatorEmail AND L.Language =
@Language),
        @RecordingLink
    );

    PRINT 'Webinar added successfully.';
END;
```

5.7. BuyWebinar

Umożliwia kupowanie webinarów przez studentów:

@StudentEmail	Email Kupującego	napis do 50 znaków
@WebinarTitle	Tytuł Webinaru	napis do 50 znaków
@Paid	Zapłacona kwota	liczba

```
CREATE PROCEDURE BuyWebinar
    @StudentEmail varchar(50),
    @WebinarTitle varchar(50),
    @Paid money = NULL
AS
BEGIN
    SET NOCOUNT ON;

    -- Declare variables
```

```
    DECLARE @StudentID int, @WebinarID int, @Price money;
    DECLARE @OrderID int;


        -- Check if the specified student exists
        IF NOT EXISTS (SELECT 1 FROM Students WHERE Email =
@StudentEmail)
        BEGIN
            PRINT 'Error: Student with email ' + @StudentEmail + '
does not exist.';
        END

        -- Check if the specified webinar exists
        IF NOT EXISTS (
            SELECT 1
            FROM Webinars AS W
            INNER JOIN Products AS P ON W.WebinarID = P.ProductID
            WHERE W.Title = @WebinarTitle
        )
        BEGIN
            PRINT 'Error: Webinar with title ' + @WebinarTitle + '
does not exist.';
        END

        -- Get StudentID
        SELECT @StudentID = StudentID FROM Students WHERE Email =
@StudentEmail;

        -- Get WebinarID and Price
        SELECT @WebinarID = W.WebinarID, @Price = P.Price
        FROM Webinars AS W
        INNER JOIN Products AS P ON W.WebinarID = P.ProductID
        WHERE W.Title = @WebinarTitle;

        -- Set @Paid to 0 if it is NULL
        IF @Paid IS NULL SET @Paid = 0;
    -- Start the transaction
    BEGIN TRANSACTION;

    BEGIN TRY
        -- Insert into Order
        INSERT INTO [Order] (StudentID, Paid, Date, PaidDate)
        VALUES (@StudentID, @Paid, GETDATE(), GETDATE());

        -- Get the OrderID of the recently inserted order
        SET @OrderID = SCOPE_IDENTITY();

        -- Insert into OrderDetails
```

```
        INSERT INTO OrderDetails (OrderID, ProductID, Price)
        VALUES (@OrderID, @WebinarID, @Price);

        -- Commit the transaction
        COMMIT;
        PRINT 'Webinar purchased successfully.';
    END TRY
    BEGIN CATCH
        -- Rollback the transaction if an error occurs
        ROLLBACK;
        PRINT 'Transaction rolled back. An error occurred during the
purchase process.';
    END CATCH;
END;
```

5.8. AddLanguage

Dodaje nowy język do Tablicy Languages:

@Language	Nazwa Języka	napis do 20 znaków

```
CREATE PROCEDURE AddLanguage
    @Language varchar(20)
AS
BEGIN
    SET NOCOUNT ON;

    -- Check if the language already exists
    IF EXISTS (SELECT 1 FROM Languages WHERE Language = @Language)
    BEGIN
        PRINT 'Error: Language ' + @Language + ' already exists.';
        RETURN;
    END

    -- Insert into Languages table
    INSERT INTO Languages (Language)
    VALUES (@Language);

    PRINT 'Language added successfully.';
END;
```

5.9. AddTranslatorLanguage

Dodaje nowy język tłumaczenia do tłumacza:

@TranslatorEmail	Email Tłumacza	napis do 50 znaków

@Language	Język tłumaczenia	napis do 20 znaków

```
CREATE PROCEDURE AddTranslatorLanguage
   @TranslatorID int,
   @Language varchar(30)
AS
BEGIN
   SET NOCOUNT ON;

   IF NOT EXISTS (SELECT * FROM Translators WHERE TranslatorID =
@TranslatorID)
   BEGIN
       PRINT 'Translator does not exist.';
       RETURN;
   END;

   IF NOT EXISTS (SELECT * FROM Languages WHERE Language =
@Language)
   BEGIN
       EXEC AddLanguage @Language;
   END;


   DECLARE @LanguageID int;
   SELECT @LanguageID = LanguageID FROM Languages WHERE Language =
@Language;

   -- Insert into TranslatorsLanguage table
   INSERT INTO TranslatorsLanguage (TranslatorID, LanguageID)
   VALUES (@TranslatorID, @LanguageID);

   PRINT 'Translator Language added successfully.';
END;
go
```

5.10. BuyCourse

Umożliwia kupowanie kursów przez studentów:

@StudentEmail	Email Studenta	napis do 50 znaków
@CourseTitle	Nazwa Kursu	napis do 50 znaków
@Paid	Zapłacona kwota	domyślnie 0

```
CREATE PROCEDURE BuyCourse
   @StudentEmail varchar(50),
   @CourseTitle varchar(50),
   @Paid money = NULL
AS
BEGIN
   SET NOCOUNT ON;

   -- Declare variables
   DECLARE @StudentID int, @CourseID int, @Price money;
   DECLARE @OrderID int;


       -- Check if the specified student exists
       IF NOT EXISTS (SELECT 1 FROM Students WHERE Email =
@StudentEmail)
       BEGIN
           PRINT 'Error: Student with email ' + @StudentEmail + '
does not exist.';
       END

       -- Check if the specified webinar exists
       IF NOT EXISTS (
           SELECT 1
           FROM Courses AS C
           INNER JOIN Products AS P ON C.CourseID = P.ProductID
           WHERE C.Title = @CourseTitle
       )
       BEGIN
           PRINT 'Error: Webinar with title ' + @CourseTitle + '
does not exist.';
       END

       -- Get StudentID
       SELECT @StudentID = StudentID FROM Students WHERE Email =
@StudentEmail;

       -- Get WebinarID and Price
       SELECT @CourseID = C.CourseID, @Price = P.Price
       FROM Courses AS C
       INNER JOIN Products AS P ON C.CourseID = P.ProductID
       WHERE C.Title = @CourseTitle;

       -- Set @Paid to 0 if it is NULL
       IF @Paid IS NULL SET @Paid = 0;
   -- Start the transaction
   BEGIN TRANSACTION;
```

```
    BEGIN TRY
        -- Insert into Order
        INSERT INTO [Order] (StudentID, Paid, Date, PaidDate)
        VALUES (@StudentID, @Paid, GETDATE(), GETDATE());

        -- Get the OrderID of the recently inserted order
        SET @OrderID = SCOPE_IDENTITY();

        -- Insert into OrderDetails
        INSERT INTO OrderDetails (OrderID, ProductID, Price)
        VALUES (@OrderID, @CourseID, @Price);

        -- Commit the transaction
        COMMIT;
        PRINT 'Course purchased successfully.';
    END TRY
    BEGIN CATCH
        -- Rollback the transaction if an error occurs
        ROLLBACK;
        PRINT 'Transaction rolled back. An error occurred during the
purchase process.';
    END CATCH;
END;
```

5.11. NewOrder

Procedura tworzy nowe zamówienie:

@StudentEmail	Email Studenta	napis do 50 znaków
@Date	Data zamówienia	domyślnie obecny dzień
@Paid	Zapłacona kwota	domyślnie 0

```
create procedure dbo.AddOrder
    @StudentEmail varchar(255),
    @Date date = NULL,
    @Paid money = NULL
as
begin
    set nocount on;

    declare @StudentID int;

    -- Assuming that Students table has an Email column
    select @StudentID = StudentID
```

```
    from dbo.Students
    where Email = @StudentEmail;

    if @Date is null set @Date = getdate();
    if @Paid is null set @Paid = 0;
    if @StudentID is not null
    begin
        insert into dbo.[Order] (StudentID, Paid, Date)
        values (@StudentID, @Paid, @Date);
    end
    else
    begin
        raiserror('Student not found for the given email.', 16, 1);
    end
```

5.12. GetSumToPay

Oblicza cenę zamówienia

@OrderID	ID zamówienia	int
@SumToPay	Zmienna przechowująca kwotę zamówienia	money, output

```
CREATE PROCEDURE dbo.GetSumToPay
    @OrderID INT,
    @SumToPay MONEY OUTPUT
AS
BEGIN
    SET NOCOUNT ON;

    SELECT @SumToPay = ISNULL(SUM(Price), 0)
    FROM dbo.OrderDetails
    WHERE OrderID = @OrderID;

    -- Print the result
    PRINT 'Sum to Pay for OrderID ' + CAST(@OrderID AS VARCHAR) + ':
' + CAST(@SumToPay AS VARCHAR);
END
```

5.13. CountStudentsOnStudy

Funkcja pozwalająca wyliczyć ilość osób zapisanych na dane studium.

@StudyID	ID studium	int
@Count	Zmienna zwracana,	int, output

	przechowująca informację o ilości studentów zapisanych na dane studium	

```
CREATE PROCEDURE CountStudentsOnStudy
   @StudyID INT,
   @Count INT OUTPUT
AS
BEGIN
   SET NOCOUNT ON;
   SELECT @Count = COUNT(*) FROM Students AS S
   WHERE S.StudentID IN(
           SELECT O.StudentID FROM [Order] AS O
           WHERE O.OrderID IN(
               SELECT OD.OrderID FROM OrderDetails AS OD
                   WHERE OD.ProductID IN(
                       SELECT P.ProductID FROM Products AS P
                       WHERE P.ProductID IN(
                           SELECT StudyID FROM Studies
                           WHERE StudyID = @StudyID
                           )
                       )
                   )
           )
END;
```

5.14. Add CountryCity

```
CREATE PROCEDURE AddCountryCity
   @Country varchar(30),
   @City varchar(30)
AS
BEGIN
   SET NOCOUNT ON;

   IF NOT @Country IN (
      SELECT CountryName FROM Countries
  )
  BEGIN
      INSERT INTO Countries (CountryName)
       VALUES (@Country)
  END

  IF NOT @City IN (
      SELECT CityName FROM Cities
```

```
  )
   BEGIN
        INSERT INTO Cities (CityName)
         VALUES (@City)
   END

   SET @Country = (SELECT CountryID FROM Countries WHERE
CountryName = @Country)
   SET @City = (SELECT CityID FROM Cities WHERE CityName = @City)

   IF NOT EXISTS (
       SELECT * FROM CountryCity
       WHERE CountryID = @Country AND CityID = @City
   )
   BEGIN
       INSERT INTO CountryCity (CountryID, CityID)
       VALUES (@Country, @City)
   END


END;
go
```

5.15. Add Course

```
CREATE PROCEDURE AddCourse
  @Title varchar(50),
  @StartingDate DATE,
  @CoordinatorID integer,
  @Price money
AS
BEGIN
  SET NOCOUNT ON;

  IF NOT @CoordinatorID IN (
      SELECT CoordinatorID FROM Coordinator
  )
  BEGIN
      PRINT 'Error: Coordinator does not exist.';
      RETURN;
  END

  IF @StartingDate < GETDATE()
   BEGIN
        PRINT 'Error: Starting date must be in the future.';
        RETURN;
   END
```

```
  DECLARE @CourseID integer;

  INSERT INTO Products(ProductTypeID, Price)
  VALUES (2, @Price)
  SET @CourseID = SCOPE_IDENTITY();
  INSERT INTO Courses (CourseID, Title, StartingDate,
CoordinatorID)
  VALUES (@CourseID, @Title, @StartingDate, @CoordinatorID);

  PRINT 'Course added successfully.';
END;
go
```

5.16. GetClasses

Funkcja pozwalająca przy pomocy widoku [Harmonogram zajęć dla studentów] dostać informację o zajęciach dla danego studenta.

@StudentID	ID studenta dla którego chcemy uzyskać informację o zajęciach	int

```
CREATE PROCEDURE GetClasses
   @StudentID INT
AS
BEGIN
   SET NOCOUNT ON;
   IF @StudentID NOT IN (SELECT StudentID FROM Students)
   BEGIN
       PRINT('Student nie istnieje.')
       RETURN
   END;

   SELECT * FROM [Harmonogram zajęć dla studentów] WHERE
     StudentID = @StudentID
END;
go
```

5.17. AddLecture

```
CREATE PROCEDURE AddLecture
  @MeetingID int,
  @SubjectTitle int,
  @Date DATETIME,
```

```
  @Lenght int,
  @Teacher varchar (50),
  @Room varchar (20) = null,
  @Link varchar (50) = Null

AS
BEGIN
  SET NOCOUNT ON;

  IF NOT @Teacher IN (
      SELECT Email FROM Teachers WHERE Email = @Teacher
  )
  BEGIN
      PRINT 'Error: Teacher does not exist.';
      RETURN;
  END

  IF @Lenght > 240
  BEGIN
      PRINT 'Lecture is too long.';
      RETURN;
  END

  SELECT @Teacher = TeacherID
  FROM Teachers
  WHERE Email = @Teacher;

  IF NOT @MeetingID IN (
      SELECT MeetingID FROM Meetings WHERE MeetingID = @MeetingID
  )
  BEGIN
      PRINT 'Error: Meeting does not exist.';
      RETURN;
  END

  SELECT @MeetingID = MeetingID
   FROM Meetings
   WHERE MeetingID = @MeetingID;

  IF NOT @SubjectTitle IN (
      SELECT SubjectName FROM Subjects WHERE SubjectName =
@SubjectTitle
  )
  BEGIN
      PRINT 'Error: Subject does not exist.';
      RETURN;
  END
```

```
  SELECT @SubjectTitle = SubjectID
   FROM Subjects
   WHERE SubjectName = @SubjectTitle;

  IF @Date < GETDATE()
   BEGIN
        PRINT 'Error: Date is in the past.';
        RETURN;
   END

   IF EXISTS (
       SELECT 1
       FROM Lecture
       WHERE MeetingID = @MeetingID AND (
           (@Date >= Lecture.Date AND @Date <= DATEADD(MINUTE,
Lecture.Length, Lecture.Date)) OR
           (Lecture.Date >= @Date AND Lecture.Date <=
DATEADD(MINUTE, @Lenght, @Date))
       )
   )
   BEGIN
       PRINT 'Error: Lecture overlaps with another module.';
       RETURN;
   END

  IF NOT @Room IN (
      SELECT RoomName FROM Room
  )
  BEGIN
      PRINT 'Error: Room does not exist.';
      RETURN;
  END

  SELECT @Room = RoomID
  FROM Room
  WHERE Room.RoomName = @Room;

   IF @Link IS NOT NULL AND @Room IS NOT NULL
   BEGIN
        PRINT 'Error: Cannot have both link and room.';
        RETURN;
   END

   IF @Link IS NULL AND @Room IS NULL
   BEGIN
        PRINT 'Error: Must have either link or room.';
        RETURN;
   END
```

```
  INSERT INTO Lecture (MeetingID, SubjectID, TeacherID, Date,
Length, RoomID, Link)
   VALUES (@MeetingID, @SubjectTitle, @Teacher, @Date, @Lenght,
@Room, @Link);

  PRINT 'Lecture added successfully.';
END;
go
```

5.18. AddMeeting

```
CREATE PROCEDURE AddMeeting
   @StudyID int,
   @Limit int,
   @Price money
AS
BEGIN
   SET NOCOUNT ON;

   IF @StudyID NOT IN (SELECT StudyID FROM Studies)
   BEGIN
       PRINT 'Study does not exist.';
       RETURN;
   END;

   IF @Limit < 1
   BEGIN
       PRINT 'Limit must be greater than 0.';
       RETURN;
   END;

   INSERT INTO Products (ProductTypeID, Price)
   VALUES (
       (SELECT ProductTypeID FROM ProductType WHERE Name =
'Meeting'),
       @Price
   );

   DECLARE @MeetingID int;
   SET @MeetingID = SCOPE_IDENTITY(); -- Get the identity value of
the recently inserted row

   INSERT INTO Meetings (MeetingID, StudyID, Limit)
   VALUES (@MeetingID, @StudyID, @Limit);
```

```
    PRINT 'Meeting added successfully.';
END;
grant execute on dbo.AddMeeting to Coordinator;
go
```

5.19. BuyMeeting

```
CREATE PROCEDURE BuyMeeting
    @StudentEmail varchar(50),
    @MeetingID varchar(50),
    @OrderID int,
    @Paid money = NULL
AS
BEGIN
    SET NOCOUNT ON;

    -- Declare variables
    DECLARE @StudentID int, @Price money;

        -- Check if the specified student exists
        IF NOT EXISTS (SELECT 1 FROM Students WHERE Email =
@StudentEmail)
        BEGIN
            PRINT 'Error: Student with email ' + @StudentEmail + '
does not exist.';
        END

        -- Check if the specified webinar exists
        IF NOT EXISTS (
            SELECT 1
            FROM Meetings AS M
            INNER JOIN Products AS P ON M.MeetingID = P.ProductID
        )
        BEGIN
            PRINT 'Error: Meeting to buy doeas not exist.';
        END

        -- Get StudentID
        SELECT @StudentID = StudentID FROM Students WHERE Email =
@StudentEmail;


        SELECT @MeetingID = M.MeetingID, @Price = P.Price
        FROM Meetings AS M
        INNER JOIN Products AS P ON M.MeetingID = P.ProductID;
```

```
        IF EXISTS (
            select 1
            FROM [Order] AS O
            inner join OrderDetails OD on O.OrderID = OD.OrderID
            where O.StudentID = @StudentID and OD.ProductID =
@MeetingID
            )
        begin
            print 'Error: Student with email ' + @StudentEmail + '
already bought this meeting.';
        end

    -- Start the transaction
    BEGIN TRANSACTION;

    BEGIN TRY
        -- Insert into OrderDetails
        INSERT INTO OrderDetails (OrderID, ProductID, Price)
        VALUES (@OrderID, @MeetingID, @Price);

        -- Commit the transaction
        COMMIT;
        PRINT 'Course purchased successfully.';
    END TRY
    BEGIN CATCH
        -- Rollback the transaction if an error occurs
        ROLLBACK;
        PRINT 'Transaction rolled back. An error occurred during the
purchase process.';
    END CATCH;
END;
go

grant execute on dbo.BuyMeeting to Student
go
```

5.20. Odrabianie

```
CREATE PROCEDURE Odrabianie
    @Studentmail varchar(30),
    @MeetindMissed int,
    @ProductToBuyID int

AS
BEGIN
    SET NOCOUNT ON;
    DECLARE @StudentID int
```

```
    SET @StudentID = (SELECT StudentID FROM Students WHERE Email =
@Studentmail)

    DECLARE @TypeOfProduct int
    SET @TypeOfProduct = (SELECT ProductTypeID FROM Products WHERE
ProductID = @ProductToBuyID)
    EXEC AddOrder @Studentmail
    DECLARE @OrderID int;
    set @OrderID = (SELECT TOP 1 OrderID FROM [Order] ORDER BY
OrderID DESC);
    IF @TypeOfProduct = 3
    BEGIN
        EXEC BuyMeeting @Studentmail, @MeetindMissed, @OrderID
    END;

    IF @TypeOfProduct = 2
    BEGIN
        EXEC BuyCourse @Studentmail, @ProductToBuyID, @OrderID
    END;

    DECLARE @Money money;
    eXEC GetSumToPay @OrderID, @Money OUTPUT;

    EXEC UpdatePaidForOrder @Studentmail, @Money, @OrderID


END;
go
```

5.21. AddModuleType

```
CREATE PROCEDURE AddModuleType
    @ModuleTypeName varchar (30)
AS
BEGIN
    SET NOCOUNT ON;

    IF @ModuleTypeName IN (
        SELECT Name FROM ModuleType
   )
   BEGIN
        PRINT 'Error: Module type already exist.';
        RETURN;
   END
```

```
  INSERT INTO ModuleType (Name)
   VALUES (@ModuleTypeName)
END;
go
```

5.22. AddModule

```
CREATE PROCEDURE AddModule
  @CourseName varchar(50),
  @Title varchar (50),
  @Date DATETIME,
  @Lenght int,
  @Teacher varchar (50),
  @Moduletype varchar (20),
  @Field varchar (30),
  @Room varchar (20) = null,
  @Link varchar (50) = Null,
  @Limit int = Null

AS
BEGIN
  SET NOCOUNT ON;

  IF NOT @Teacher IN (
      SELECT Email FROM Teachers WHERE Email = @Teacher
  )
  BEGIN
      PRINT 'Error: Teacher does not exist.';
      RETURN;
  END

  IF @Lenght > 240
  BEGIN
      PRINT 'Module is too long.';
      RETURN;
  END

  SELECT @Teacher = TeacherID
  FROM Teachers
  WHERE Email = @Teacher;

  IF NOT @CourseName IN (
      SELECT Title FROM Courses WHERE Title = @CourseName
  )
  BEGIN
      PRINT 'Error: Course does not exist.';
```

```
      RETURN;
  END

  SELECT @CourseName = CourseID
  FROM Courses
  WHERE Title = @CourseName;

  IF @Date < GETDATE()
   BEGIN
        PRINT 'Error: Date is in the past.';
        RETURN;
   END

   IF EXISTS (
       SELECT 1
       FROM Modules
       WHERE CourseID = @CourseName
       AND (
           (@Date >= Modules.Date AND @Date <= DATEADD(MINUTE,
Modules.Length, Modules.Date)) OR
           (Modules.Date >= @Date AND Modules.Date <=
DATEADD(MINUTE, @Lenght, @Date))
       )
   )
   BEGIN
       PRINT 'Error: Module overlaps with another module.';
       RETURN;
   END

  IF NOT @Moduletype IN (
      SELECT Name FROM ModuleType
  )
  BEGIN
        PRINT 'Error: Module type does not exist.';
        RETURN;
  END

  SELECT @Moduletype = ModuleTypeID
  FROM ModuleType
  WHERE ModuleType.Name = @Moduletype;

  IF NOT @Room IN (
      SELECT RoomName FROM Room
  )
  BEGIN
      PRINT 'Error: Room does not exist.';
      RETURN;
  END
```

```
  SELECT @Room = RoomID
  FROM Room
  WHERE Room.RoomName = @Room;

  IF EXISTS (
       SELECT 1
       FROM Modules
       WHERE RoomID = @Room AND (
           (@Date >= Modules.Date AND @Date <= DATEADD(MINUTE,
Modules.Length, Modules.Date)) OR
           (Modules.Date >= @Date AND Modules.Date <=
DATEADD(MINUTE, @Lenght, @Date))
       )
   )
   BEGIN
       PRINT 'Error: Room is taken module.';
       RETURN;
   END

   IF EXISTS (
       SELECT 1
       FROM Lecture
       WHERE RoomID = @Room AND (
           (@Date >= Lecture.Date AND @Date <= DATEADD(MINUTE,
Lecture.Length, Lecture.Date)) OR
           (Lecture.Date >= @Date AND Lecture.Date <=
DATEADD(MINUTE, @Lenght, @Date))
       )
   )
   BEGIN
       PRINT 'Error: Room is taken module.';
       RETURN;
   END

  IF NOT @Field IN (
      SELECT Name FROM FieldType
  )
  BEGIN
      INSERT INTO FieldType (Name) VALUES (@Field);
  END

  SELECT @Field = FieldTypeID
  FROM FieldType
  WHERE FieldType.Name = @Field;

  IF @Limit IS NOT NULL AND @Limit < 5
   BEGIN
```

```
        PRINT 'Error: Limit is too low.';
        RETURN;
    END

    IF @Link IS NOT NULL AND @Room IS NOT NULL
    BEGIN
        PRINT 'Error: Cannot have both link and room.';
        RETURN;
    END

    IF @Link IS NULL AND @Room IS NULL
    BEGIN
        PRINT 'Error: Must have either link or room.';
        RETURN;
    END

    IF @Moduletype IN (2, 3) AND (@Link IS NULL OR @Room IS NOT
NULL)
    BEGIN
        PRINT 'Error: online module Link must be set and room must
be null.';
        RETURN;
    END

    IF @Moduletype = 1 AND (@Link IS NOT NULL OR @Room IS NULL)
    BEGIN
        PRINT 'Error: in-person module must be null and room must
be set.';
        RETURN;
    END

  INSERT INTO Modules (CourseID, Title, Date, Length, TeacherID,
ModuleType, Link, Limit, FieldID)
  VALUES (@CourseName, @Title, @Date, @Lenght, @Teacher,
@Moduletype, @Link, @Limit, @Field);

  PRINT 'Module added successfully.';
END;
grant execute on dbo.AddModule to Coordinator;
go
```

6. Triggery

6.1. Aktualizacja widoku ‘Studenci i zdawanie przedmiotów’

```
CREATE TRIGGER trg_UpdateAttendanceAndGrade
ON dbo.LecturePresence
AFTER INSERT
AS
BEGIN
    SET NOCOUNT ON;
    UPDATE av
    SET
        [Percent] = a.[Percent],
        AverageGrade = ag.AverageGrade,
        [Status] = CASE WHEN a.[Percent] >= 0.8 AND ag.AverageGrade
>= 3 THEN 'Pass' ELSE 'Fail' END
    FROM
        dbo.AttendanceAndGradeView av
        INNER JOIN inserted i ON av.StudentID = i.StudentID
        LEFT JOIN (
            SELECT
                LP.StudentID,
                L.SubjectID,
                SUM(CAST(LP.WasPresent AS int)) * 1.0 / COUNT(*) AS
[Percent]
            FROM
                inserted i
                INNER JOIN LecturePresence LP ON i.LectureID =
LP.LectureID
                INNER JOIN Lecture L ON LP.LectureID = L.LectureID
            GROUP BY
                LP.StudentID,
                L.SubjectID
        ) a ON av.StudentID = a.StudentID AND av.SubjectID =
a.SubjectID
        LEFT JOIN (
            SELECT
                StudentID,
                SubjectID,
                AVG(Grade) AS AverageGrade
            FROM
                SubjectGrade
            GROUP BY
                StudentID,
                SubjectID
```

```
        ) ag ON av.StudentID = ag.StudentID AND av.SubjectID =
ag.SubjectID;
END
```

6.2. Zaliczanie kursów na podstawie obecności

```
CREATE TRIGGER TrgPassedCourse
ON ModulePresence
AFTER INSERT, UPDATE, DELETE
AS
BEGIN
   DECLARE @PresenceID int
   SET @PresenceID = (SELECT MAX(ModulePresenceID) FROM
ModulePresence)
   DECLARE @ModuleID int
   SET @ModuleID = (SELECT ModuleID FROM ModulePresence WHERE
ModulePresenceID = @PresenceID)
   DECLARE @StudentID int
   SET @StudentID = (SELECT StudentID FROM ModulePresence WHERE
ModulePresenceID = @PresenceID)
   DECLARE @CourseID int
   SET @CourseID = (SELECT CourseID FROM Modules WHERE ModuleID =
@ModuleID)
   DECLARE @Course_modules int
   SET @Course_modules = (SELECT COUNT(ModuleID) FROM Modules WHERE
CourseID = @CourseID)
   IF (SELECT COUNT(Modules.ModuleID) FROM Modules
       INNER JOIN ModulePresence ON Modules.ModuleID =
ModulePresence.ModuleID
       WHERE ModulePresence.StudentID = @StudentID AND
Modules.CourseID = @CourseID) > 0.8 * @Course_modules
   BEGIN
       INSERT INTO passedCourse (StudentID, CourseID) VALUES
(@StudentID, @CourseID)
   END
   end
go
```

6.3. Zaliczanie studiów na podstawie praktyk i ocen

```
CREATE TRIGGER trg_CheckPassedStudies
ON dbo.Grades
AFTER INSERT
AS
BEGIN
   SET NOCOUNT ON;
```

```
    DECLARE @studyId INT, @userId INT, @subjectId INT,
@numberOfGrades INT;
    set @subjectId = (SELECT SubjectID fROM inserted);
    set @studyId = (SELECT StudyID
                    FROM Subjects
                    WHERE Subjects.SubjectID = @subjectId);
    set @userId = (SELECT StudentID FROM inserted);
    set @numberOfGrades = (SELECT COUNT(*) FROM Subjects where
StudyID = @studyId);
    if not exists (
        select 1
        from Practises
        where StudyID = @studyId and StudentID = @userId
    )
    Begin
        return;
    end

    --compare num of grades with num of grades in subject
    if @numberOfGrades > (SELECT COUNT(*)
                          FROM Grades
                          left join Subjects on Grades.SubjectID =
Subjects.SubjectID
                          where StudyID = @studyId and StudentID =
@userId)
    Begin
        return;
    end

    if exists (
        select 1
        from Grades
        left join Subjects on Grades.SubjectID = Subjects.SubjectID
        where StudyID = @studyId and StudentID = @userId and Grade <
3
    )
    Begin
        return;
    end
    Begin
        Insert Into PassedStudies (StudyID, userId) Values
(@studyId, @userId);
    end
END;
```

6.4. Zaliczenie kursu na podstawie obecności

```
CREATE TRIGGER TrgPassedCourse
```

```
ON ModulePresence
AFTER INSERT, UPDATE, DELETE
AS
BEGIN
   DECLARE @PresenceID int
   SET @PresenceID = (SELECT MAX(ModulePresenceID) FROM
ModulePresence)
   DECLARE @ModuleID int
   SET @ModuleID = (SELECT ModuleID FROM ModulePresence WHERE
ModulePresenceID = @PresenceID)
   DECLARE @StudentID int
   SET @StudentID = (SELECT StudentID FROM ModulePresence WHERE
ModulePresenceID = @PresenceID)
   DECLARE @CourseID int
   SET @CourseID = (SELECT CourseID FROM Modules WHERE ModuleID =
@ModuleID)
   DECLARE @Course_modules int
   SET @Course_modules = (SELECT COUNT(ModuleID) FROM Modules WHERE
CourseID = @CourseID)
   IF (SELECT COUNT(Modules.ModuleID) FROM Modules
       INNER JOIN ModulePresence ON Modules.ModuleID =
ModulePresence.ModuleID
       WHERE ModulePresence.StudentID = @StudentID AND
Modules.CourseID = @CourseID) > 0.8 * @Course_modules
   BEGIN
       INSERT INTO passedCourse (StudentID, CourseID) VALUES
(@StudentID, @CourseID)
   END
   end
```

7. Indeksy

```
create index CountryCity_CountryCityID_CountryID_CityID_index
   on dbo.CountryCity (CountryCityID, CountryID, CityID)
```

```
create index Courses_CourseID_Title_index
   on Courses (CourseID, Title)
```

```
create index FieldType_FieldTypeID_Name_index
   on FieldType (FieldTypeID, Name)
```

```
create index Grades_GradeID_SubjectID_StudentID_index
   on Grades (GradeID, SubjectID, StudentID)
```

```
create index
LecturePresence_LecturePresenceID_StudentID_LectureID_index
   on LecturePresence (LecturePresenceID, StudentID, LectureID)
```

```
create index Modules_ModuleID_ModuleType_index
   on Modules (ModuleID, ModuleType)
```

```
create index OrderDetails_OrderDetailsID_OrderID_ProductID_index
   on OrderDetails (OrderDetailsID, OrderID, ProductID)
```

```
create index Products_ProductID_ProductTypeID_index
   on Products (ProductID, ProductTypeID)
```

```
create index Studies_StudyID_CoordinatorID_index
   on Studies (StudyID, CoordinatorID)
```

8. Uprawnienia

8.1. Tłumacz

```
use u_wojtas
go

create role Teacher authorization dbo
go

grant delete, insert, update on dbo.LecturePresence to Teacher
go

grant delete, insert, update on dbo.ModulePresence to Teacher
go

grant execute on dbo.AddGrade to Teacher
go
```

8.2. Starszy Administrator

```

create role HeadAdministrator authorization dbo
go

grant alter any role, create function, create procedure, create
table, create view, execute on database :: u_wojtas to
HeadAdministrator
go

grant delete, insert, select, update on dbo.AttendanceAndGrade to
HeadAdministrator
go

grant alter, control, delete, insert, select, update on dbo.Cities
to HeadAdministrator
go

grant delete, insert, select, update on dbo.Coordinator to
HeadAdministrator
go
```

```
grant delete, insert, select, update on dbo.Countries to
HeadAdministrator
go

grant delete, insert, select, update on dbo.Courses to
HeadAdministrator
go

grant delete, insert, select, update on dbo.FieldType to
HeadAdministrator
go

grant delete, insert, select, update on dbo.Grades to
HeadAdministrator
go

grant delete, insert, select, update on dbo.Languages to
HeadAdministrator
go

grant delete, insert, select, update on dbo.Lecture to
HeadAdministrator
go

grant delete, insert, select, update on dbo.LecturePresence to
HeadAdministrator
go

grant delete, insert, select, update on dbo.Meetings to
HeadAdministrator
go

grant delete, insert, select, update on dbo.ModulePresence to
HeadAdministrator
go

grant delete, insert, select, update on dbo.Modules to
HeadAdministrator
go

grant delete, insert, select, update on dbo.[Order] to
HeadAdministrator
go

grant delete, insert, select, update on dbo.OrderDetails to
HeadAdministrator
go
```

```
grant delete, insert, select, update on dbo.Postponed to
HeadAdministrator
go

grant delete, insert, select, update on dbo.Practises to
HeadAdministrator
go

grant delete, insert, select, update on dbo.ProductType to
HeadAdministrator
go

grant delete, insert, select, update on dbo.Products to
HeadAdministrator
go

grant delete, insert, select, update on dbo.Room to
HeadAdministrator
go

grant delete, insert, select, update on dbo.Students to
HeadAdministrator
go

grant delete, insert, select, update on dbo.Studies to
HeadAdministrator
go

grant delete, insert, select, update on dbo.SubjectGrade to
HeadAdministrator
go

grant delete, insert, select, update on dbo.Subjects to
HeadAdministrator
go

grant delete, insert, select, update on dbo.Teachers to
HeadAdministrator
go

grant delete, insert, select, update on dbo.Translators to
HeadAdministrator
go

grant delete, insert, select, update on dbo.TranslatorsLanguage to
HeadAdministrator
go
```

```
grant delete, insert, select, update on dbo.Webinars to
HeadAdministrator
go

grant delete, insert, select, update on dbo.passedCourse to
HeadAdministrator
go

grant delete, insert, select, update on dbo.passedStudies to
HeadAdministrator
go

grant alter on [Microsoft.SqlServer.Types] to HeadAdministrator
go
```

8.3. Młodszy Administrator

```
create role Administrator authorization dbo
go

grant create function, create procedure, create view, execute on
database :: u_wojtas to Administrator
go

grant delete, insert, select, update on dbo.Cities to Administrator
go

grant delete, insert, select, update on dbo.Coordinator to
Administrator
go

grant delete, insert, select, update on dbo.Countries to
Administrator
go

grant delete, insert, select, update on dbo.Courses to Administrator
go

grant delete, insert, select, update on dbo.FieldType to
Administrator
go

grant delete, insert, select, update on dbo.Grades to Administrator
go

grant delete, insert, select, update on dbo.Languages to
Administrator
```

```
go

grant delete, insert, select, update on dbo.Lecture to Administrator
go

grant delete, insert, select, update on dbo.LecturePresence to
Administrator
go

grant delete, insert, select, update on dbo.Meetings to
Administrator
go

grant delete, insert, select, update on dbo.ModulePresence to
Administrator
go

grant delete, insert, select, update on dbo.Modules to Administrator
go

grant delete, insert, select, update on dbo.[Order] to Administrator
go

grant delete, insert, select, update on dbo.OrderDetails to
Administrator
go

grant delete, insert, select, update on dbo.Postponed to
Administrator
go

grant delete, insert, select, update on dbo.Practises to
Administrator
go

grant delete, insert, select, update on dbo.ProductType to
Administrator
go

grant delete, insert, select, update on dbo.Products to
Administrator
go

grant delete, insert, select, update on dbo.Room to Administrator
go

grant delete, insert, select, update on dbo.Students to
Administrator
```

```
go

grant delete, insert, select, update on dbo.Studies to Administrator
go

grant delete, insert, select, update on dbo.SubjectGrade to
Administrator
go

grant delete, insert, select, update on dbo.Subjects to
Administrator
go

grant delete, insert, select, update on dbo.Teachers to
Administrator
go

grant delete, insert, select, update on dbo.Translators to
Administrator
go

grant delete, insert, select, update on dbo.TranslatorsLanguage to
Administrator
go

grant delete, insert, select, update on dbo.Webinars to
Administrator
go

grant delete, insert, select, update on dbo.passedCourse to
Administrator
go

grant delete, insert, select, update on dbo.passedStudies to
Administrator
go
```

8.4. Koordynator

```
create role Coordinator authorization dbo
go

grant delete, insert, select, update on dbo.Coordinator to
Coordinator
go
```

```
grant delete, insert, select, update on dbo.Courses to Coordinator
go

grant delete, insert, select, update on dbo.FieldType to Coordinator
go

grant delete, insert, select, update on dbo.Grades to Coordinator
go

grant delete, insert, select, update on dbo.Lecture to Coordinator
go

grant delete, insert, select, update on dbo.LecturePresence to
Coordinator
go

grant delete, insert, select, update on dbo.Meetings to Coordinator
go

grant delete, insert, select, update on dbo.ModulePresence to
Coordinator
go

grant delete, insert, select, update on dbo.Modules to Coordinator
go

grant delete, insert, select, update on dbo.Room to Coordinator
go

grant delete, insert, select, update on dbo.Studies to Coordinator
go

grant delete, insert, select, update on dbo.SubjectGrade to
Coordinator
go

grant delete, insert, select, update on dbo.Subjects to Coordinator
go

grant delete, insert, select, update on dbo.Webinars to Coordinator
go

grant delete, insert, select, update on dbo.passedCourse to
Coordinator
go

grant delete, insert, select, update on dbo.passedStudies to
Coordinator
```

```
go

grant execute on dbo.AddCourse to Coordinator
go

grant execute on dbo.AddGrade to Coordinator
go

grant execute on dbo.AddModule to Coordinator
go
```

8.5. Dyrektor

```
create role Director authorization dbo
go

grant delete, insert, select, update on dbo.Postponed to Director
go

grant select on dbo.[Frekwencja na zakonczonych wydarzeniach - done]
to Director
go

grant select on dbo.[Harmonogram zajęć dla studentów] to Director
go

grant select on dbo.[Lista dluznikow - done] to Director
go

grant select on dbo.[Lista obecnosci dla kazdego szkolenia - done]
to Director
go

grant select on dbo.[Opis kursu] to Director
go

grant select on dbo.[Osoby na wydarzeniach -done] to Director
go

grant select on dbo.Praktyki to Director
go

grant select on dbo.[Przedmioty posiadane przez użytkowników] to
Director
go

grant select on dbo.[Raport Bilokacji - done] to Director
```

```
go

grant select on dbo.[Raporty finansowe - done] to Director
go

grant select on dbo.[Spis pracowników] to Director
go

grant select on dbo.[Syllabus studiów] to Director
go
```

8.6. Student

```
create role Student authorization dbo
go

grant execute on dbo.BuyCourse to Student
go

grant execute on dbo.BuyStudy to Student
go

grant execute on dbo.BuyWebinar to Student
go

grant execute on dbo.GetClasses to Student
go
```